Information and Computation **149**, 42–76 (1999)
Article ID inco.1998.2751, available online at http://www.idealibrary.com on IDEAL®

Order-Sorted Inductive Types

Gilles Barthe

Departamento de Informática, Universidade do Minho, 4710 Braga, Portugal
E-mail: gilles@di.uminho.pt

System $F^{\omega}_{\leqslant}$ is an extension of system F^{ω} with subtyping and bounded quantification. Order-sorted algebra is an extension of many-sorted algebra with overloading and subtyping. We combine both formalisms to obtain $IF^{\omega}_{\leqslant}$, a higher-order typed λ-calculus with subtyping, bounded quantification, and order-sorted inductive types, i.e., data types with built-in subtyping and overloading. Moreover we show that $IF^{\omega}_{\leqslant}$ enjoys important meta-theoretic properties, including confluence, strong normalization, subject reduction, and decidability of type checking. © 1999 Academic Press

1. INTRODUCTION

Typed functional programming languages such as Haskell and ML and type-theory-based proof-development systems such as Coq and Lego support the introduction of *inductively defined types* such as natural numbers or booleans, parameterized inductively defined types such as lists, and even parameterized mutual inductively defined types such as trees and forests. In addition, those languages support the definition of functions by pattern matching or by recursion, and in the case of proof-development systems also of a mechanism to prove properties by induction. Such inductive definitions constitute a fundamental ingredient in the expressivity of these systems; in fact, one can argue that inductive definitions, together with λ-calculus, provide the core of these systems.

Subtyping and *overloading* are powerful abstractions that permeate through computer science. Their relevance to programming languages has long been recognized, in particular by Goguen and Meseguer in their work on order-sorted algebra (OSA) [GM92]. The basic concept of OSA. is that of the order-sorted signature, which extends the traditional notion of the many-sorted signature with subtyping and overloading (in fact, the latter is already present in MSA [GM85]). More precisely, order-sorted signatures extend their many-sorted counterpart with a new mechanism to declare a type A as a subtype of another type B, typically,

$$\mathsf{odd} \leqslant \mathsf{nat}, \qquad \mathsf{even} \leqslant \mathsf{nat}$$

0890-5401/99 $30.00

and allow for function symbols to be overloaded, i.e., assigned more than one domain and codomain, typically,

$$s : \mathsf{nat} \to \mathsf{nat}$$
$$s : \mathsf{even} \to \mathsf{odd}$$
$$s : \mathsf{odd} \to \mathsf{even}$$

or

$$+ : \mathsf{nat} \to \mathsf{nat} \to \mathsf{nat}$$
$$+ : \mathsf{odd} \to \mathsf{odd} \to \mathsf{even}$$
$$+ : \mathsf{even} \to \mathsf{even} \to \mathsf{even}$$
$$+ : \mathsf{even} \to \mathsf{odd} \to \mathsf{odd}$$
$$+ : \mathsf{odd} \to \mathsf{even} \to \mathsf{odd}.$$

Subtyping is embodied in the resulting language of terms via a new rule, called subsumption, that turns every inhabitant of A into an inhabitant of B. In the above example, the subsumption rule is instantiated to the two rules

$$\frac{t : \mathsf{odd}}{t : \mathsf{nat}}, \qquad \frac{t : \mathsf{even}}{t : \mathsf{nat}}.$$

The combination of subtyping and overloading yields a concise and readable framework for describing data types in terms of their constructors. However, order-sorted data types, i.e., data types specified by order-sorted signatures, do not always support recursive definitions. Indeed, order-sorted data types are inherently non-deterministic constructions: in the above example, one may derive $s0 : \mathsf{nat}$ from $0 : \mathsf{even}$ in two different ways:

$$\frac{\dfrac{0 : \mathsf{even}}{0 : \mathsf{nat}}}{s0 : \mathsf{nat}} \qquad \frac{\dfrac{0 : \mathsf{even}}{s0 : \mathsf{odd}}}{s0 : \mathsf{nat}}$$

In that case, non-determinism is caused by subtyping. Another example where non-determinism is caused by overloading is given by a specification with three sorts σ, τ, v with a constant c of type σ and τ and unary function symbol $f : \sigma \to v$ and $f : \tau \to v$. Again, one may derive $f\,c : v$ in two different ways:

$$\frac{c : \sigma}{f\,c : v}, \qquad \frac{c : \tau}{f\,c : v}.$$

For such signatures, it is not obvious *a priori* what is a correct case-expression nor how such case-expressions should be evaluated (the same remarks apply to recursive definitions). The problem is that non-determinism induces a conflict when

trying to evaluate case-expressions. If we adopt the usual convention that case-expressions should contain one branch for each constructor declaration, then a case-expression over v would take the form

$$\begin{aligned}&\mathsf{case}_v\ t\ \mathsf{of}\ (f\,x) \Rightarrow l \quad |\\ &\qquad\qquad (f\,x) \Rightarrow r.\end{aligned}$$

The intended meaning of such an expression is that it should evaluate to $l[t'/x]$ if $t = f\,t'$ and $t' : \sigma$ or to $r[t'/x]$ if $t = f\,t'$ and $t' : \tau$. However, these evaluation rules are ambiguous if $t = f\,c$ since $c : \sigma$ and $c : \tau$. Of course, one could adopt the convention, usual in functional programming, that the first rule to apply determines the meaning of the expression but this solution is contrived and threatens confluence of the reduction calculus.

A simpler solution is to restrict overloading so as to rule out conflictual situations such as the one described above. A first contribution of the paper is to isolate the class of *strictly overloaded* order-sorted signatures, a large class of signatures which admit a well-behaved theory of recursive definitions. This class includes many order-sorted signatures of interest, e.g., that of natural numbers in Table 1.

The second and main contribution of the paper is the definition and study of $IF^{\omega}_{\leqslant}$, a higher-order, typed λ-calculus combining subtyping, bounded quantification, and order-sorted inductive types. The inductive core of $IF^{\omega}_{\leqslant}$ is given by

TABLE 1

Case-Expressions for a Strictly Overloaded Signature

Odd and even natural numbers

Sorts:	even odd nat
Subsort relation:	even, off $\leqslant$ nat
Declarations:	0 : even
	s : even $\to$ odd
	s : odd $\to$ even
	s : nat $\to$ nat

There is no ambiguity in defining case-expressions:

- case-expression over nat:

$$\begin{aligned}&\mathsf{case}_{\mathsf{nat}}\ t\ \mathsf{of} \quad 0 \Rightarrow l \quad |\\ &\qquad\qquad\quad (s\,x) \Rightarrow r\end{aligned}$$

- case-expression over even:

$$\begin{aligned}&\mathsf{case}_{\mathsf{even}}\ t\ \mathsf{of} \quad 0 \Rightarrow l \quad |\\ &\qquad\qquad\quad (s\,x) \Rightarrow r\end{aligned}$$

- case-expression over odd:

$$\mathsf{case}_{\mathsf{odd}}\ t\ \mathsf{of} \quad (s\,x) \Rightarrow b$$

with the expected reduction rules.

strictly overloaded order-sorted data types together with accompanying mechanisms for recursive definitions. The λ-calculus core is given by Cardelli's $F^{\omega}_{\leqslant}$ [Car90], a typed λ-calculus with subtyping and bounded quantification that provides a theoretical model of object-oriented programming [HP95, PT94, PS97]. We show that $IF^{\omega}_{\leqslant}$ enjoys important meta-theoretic properties, including subject reduction, strong normalization, and decidability of type checking.

This work is motivated by a perceived need to enhance typed functional programming languages and proof-development systems with subtyping; see, e.g., [AC96b, FP91, Hal93, Luo98, Pfe93]. By addressing the issue of inductively defined types in a typed λ-calculus with subtyping, we hope to contribute toward the integration of subtyping in typed functional languages and in proof-development systems. In this respect, our choice of $F^{\omega}_{\leqslant}$ is dictated by a series of proposals for the language to serve as a basis for new programming languages combining functional and object-oriented features; see, e.g, [AC96a].

The paper is organized as follows: Section 2 provides a brief introduction to order-sorted algebra. In Section 3, we study recursive definitions in an order-sorted setting. Section 4 introduces system $IF^{\omega}_{\leqslant}$. Its properties are studied in Section 5. In Section 6, we conclude with related work and directions for further research.

Notation. For every set A, we let A^{ω} denote the set of lists over A, [ ] denote the empty list, :: denote the usual cons-operation, $\in$ denote list membership, $\#l$ denote the length of $l \in A^{\omega}$, and $l[i]$ denote, when it exists, the ith element of l. For convenience, we will sometimes write lists in the form $(a_1, ..., a_n)$ instead of $a_1 :: \cdots :: a_n :: [\ ]$.

Throughout the paper, we shall adopt some conventions for finite maps. If $X = \{x_1, ..., x_n\}$ is a finite set and A is an X-indexed family of sets, then a function $f \in \prod \alpha \in X \cdot A_{\alpha}$ is specified by providing the images $a_1 \in A_{x_1}, ..., a_n \in A_{x_n}$ of $x_1, ..., x_n$, respectively. In some circumstances, we may write $f = [x_1 \mapsto a_1, ..., x_n \mapsto a_n]$. Moreover, we let f_x denote the application of $f \in \prod \alpha \in X \cdot A_{\alpha}$ to $x \in X$.

Finally, we let **Set** denote the class of sets.

2. ORDER-SORTED ALGEBRA

In order to introduce parameterized inductive types such as the type of lists over an arbitrary type, we consider a variant of order-sorted signatures which distinguishes between sorts and parameters. The distinction becomes meaningful only when combining order-sorted algebra with a type system: in this context, sorts will be treated as constants whereas parameters will be treated as variables.

DEFINITION 1. A *sort structure* consists of a pair $(\mathscr{P}, (\Lambda, \leqslant))$, where $\mathscr{P}$ is a finite set of *parameters* and $(\Lambda, \leqslant)$ is a finite partial order of *sorts*.

For the sake of hygiene, we always assume that $\Lambda \cap \mathscr{P} = \varnothing$. Note that $\Lambda \cup \mathscr{P}$ is partially ordered by the disjoint union $\leqslant^{sp}$ of $\leqslant$ and the set-theoretic equality $=$ on $\mathscr{P}$ and that $(\Lambda \cup \mathscr{P})^{\omega}$ is partially ordered by the componentwise extension $(\leqslant^{sp})^{\omega}$ of $\leqslant^{sp}$. In the following, we often drop the subscripts and write $\leqslant$ instead of $\leqslant^{sp}$ or $\leqslant^{\omega}$.

DEFINITION 2 (Signature). 1. A *signature* Σ over a sort structure $(\mathscr{P}, (\Lambda, \leqslant))$ consists of a finite set $\mathscr{F}$ of *function symbols* and of a function decl that assigns to each function symbol $f \in \mathscr{F}$ a finite set of *declarations* of the form

$$f : \boldsymbol{\sigma} \to \tau,$$

where $\boldsymbol{\sigma} \in (\Lambda \cup \mathscr{P})^{\omega}$ and $\tau \in \Lambda$.

2. For every $\tau \in \Lambda$, $n \in \mathbb{N}$, and $f \in \mathscr{F}$, the set $\mathsf{Dom}^{\Sigma}_{(\tau, n, f)}$ of n-ary τ-domains of f is defined as

$$\mathsf{Dom}^{\Sigma}_{(\tau, n, f)} = \{\boldsymbol{\sigma} \in (\Lambda \cup \mathscr{P})^{\omega} \mid \#\boldsymbol{\sigma} = n \wedge \exists \tau' \in \Lambda \cdot \tau' \leqslant \tau \wedge f : \boldsymbol{\sigma} \to \tau'\}.$$

3. For every $\boldsymbol{\sigma} \in (\Lambda \cup \mathscr{P})^{\omega}$ and $f \in \mathscr{F}$, the set $\mathsf{Codom}^{\Sigma}_{(\boldsymbol{\sigma}, f)}$ of *codomains* of f is defined as

$$\mathsf{Codom}^{\Sigma}_{(\boldsymbol{\sigma}, f)} = \{\tau \in \Lambda \mid \exists \boldsymbol{\sigma}' \in (\Lambda \cup \mathscr{P})^{\omega} \cdot \boldsymbol{\sigma} \leqslant \boldsymbol{\sigma}' \wedge f : \boldsymbol{\sigma}' \to \tau\}.$$

4. A triple $(\tau, n, f) \in \Lambda \times \mathbb{N} \times \mathscr{F}$ is *relevant* if $\mathsf{Dom}^{\Sigma}_{(\tau, n, f)} \neq \varnothing$. The set of relevant triples is denoted by $\mathscr{R}$.

Our definition of signature is very liberal, as we impose no restriction on the possible typings of function symbols; see [GD94, GM92] for alternative notions of signatures. Some examples of signatures may be found in Tables 2 and 3: the signature of integers, the parametric signature of lists with non-empty lists, the signature of Harrop formulae (this example is adapted from [Pfe93], note how overloading eliminates the need for intersection types), and the signature of an object calculus (this example is inspired by [AC96a] but we do not require labels to be assigned at most once in an object; our handling of free and bound variables, which distinguishes between parameters and variables, follows [Pol94]).

TABLE 2

Examples of Signatures, 1

	Integers
Parameters:	
Sorts:	pos neg int
Subsort relation:	pos, neg $\leqslant$ int
Declarations:	0 : pos
	0 : neg
	s : pos $\to$ pos
	p : neg $\to$ neg
	Lists
Parameters:	p
Sorts:	NeList List
Subsort relation:	NeList $\leqslant$ List
Declarations:	nil : List
	cons : List, $p \to$ NeList

TABLE 3

Examples of Signatures, 2

Harrop formulae

Parameters:	atom
Sorts:	program goal formula
Subsort relation:	program, goal $\leqslant$ formula
Declarations:	i : atom $\rightarrow$ goal
	i : atom $\rightarrow$ program
	$\wedge$: goal $\rightarrow$ goal $\rightarrow$ goal
	$\wedge$: program $\rightarrow$ program $\rightarrow$ program
	$\wedge$: formula $\rightarrow$ formula $\rightarrow$ formula
	$\vee$: goal $\rightarrow$ goal $\rightarrow$ goal
	$\vee$: formula $\rightarrow$ formula $\rightarrow$ formula
	$\supset$: goal $\rightarrow$ program $\rightarrow$ program
	$\supset$: program $\rightarrow$ goal $\rightarrow$ goal
	$\supset$: formula $\rightarrow$ formula $\rightarrow$ formula

Object calculus

Parameters:	lab, par, var
Sorts:	exp obj met
Subsort relation:	obj $\leqslant$ exp
Declarations:	$\square$: obj
	extobj : obj, met $\rightarrow$ obj
	i : par $\rightarrow$ exp
	i' : var $\rightarrow$ exp
	sel : obj, lab $\rightarrow$ exp
	upd : obj, met $\rightarrow$ exp
	app : exp, exp $\rightarrow$ exp
	abs : exp, var $\rightarrow$ exp
	ζ : exp, var, lab $\rightarrow$ met

OSA generalizes several well-established formalisms that have appeared in the literature: many-sorted algebra, single-sorted algebra,.... Each such formalism considers a subclass of OSA signatures. The next definition introduces some of the most important such subclasses.

DEFINITION 3. Let Σ be a signature over $(\mathscr{P}, (\Lambda, \leqslant))$.

1. Σ is *parametric* if $\mathscr{P} \neq \varnothing$.

2. Σ is *overloaded* if there exists a function symbol f that is multiply declared, i.e., such that $\mathsf{decl}(f)$ contains more than one element.

3. Σ is *single-sorted* if Λ is a singleton.

4. Σ is *many-sorted* if $\leqslant$ coincides with the set-theoretic equality on Λ.

We conclude this section with a syntactic construction of initial algebras. The construction distinguishes between sorts and parameters: the latter are instantiated as sets and the former are built inductively from those sets.

DEFINITION 4. Let $(\mathscr{P}, (\Lambda, \leqslant))$ be a sort structure, let $\Sigma = (\mathscr{F}, \mathsf{decl})$ be a signature over $(\mathscr{P}, (\Lambda, \leqslant))$, and let $I \in \mathscr{P} \to \mathbf{Set}$. A Σ-*term* of sort/parameter τ over I is an expression t s.t. $|\vdash_\Sigma^I t : \sigma$ is derivable from the following rules:

$$(Var) \qquad \frac{}{|\vdash_\Sigma^I x : p} \qquad \text{if} \quad x \in I(p)$$

$$(Sub) \qquad \frac{|\vdash_\Sigma^I t : \sigma}{|\vdash_\Sigma^I t : \tau} \qquad \text{if} \quad \sigma \leqslant \tau$$

$$(Fun) \qquad \frac{|\vdash_\Sigma^I t_1 : \sigma \cdots |\vdash_\Sigma^I t_n : \sigma_n}{|\vdash_\Sigma^I f(t_1, ..., t_n) : \tau} \qquad \text{if} \quad f : (\sigma_1, ..., \sigma_n) \to \tau.$$

The set of Σ-terms of sort/parameter τ over I is denoted by $\mathscr{T}_\Sigma^I(\tau)$. Finally, for $\boldsymbol{\tau} = (\tau_1, ..., \tau_n) \in (\Lambda \cup \mathscr{P})^\omega$, we let $\mathscr{T}_\Sigma^I(\boldsymbol{\tau})$ denote $(\mathscr{T}_\Sigma^i(\tau_1), ..., \mathscr{T}_\Sigma^i(\tau_n))$.

3. RECURSIVE DEFINITIONS

Recursion is a well-understood concept in the single-sorted case but is problematic in the order-sorted case. In this section, we review the principles subsumed by recursion and develop a theory of order-sorted recursive definitions. For the clarity of the exposition, we view recursion as the combination of two principles:

1. a *definitional* principle, allowing for functions to be defined recursively;
2. a *computational* principle, specifying the behavior of recursively defined functions.

We begin by exemplifying (in a set-theoretic framework) those principles with single-sorted and non-parametric signatures before proceeding gradually toward order-sorted, parameterized signatures.

3.1. Non-parametric Single-Sorted Signatures

Natural numbers provide an archetypical example of inductively defined type. Syntactically, the natural numbers may be seen as the set of terms of the single-sorted signature with one constant 0 and one unary function s. In that context, the definitional principle states that for every set A, every pair (f_0, f_s) where $f_0 \in A$ and $f_s \in \mathsf{nat} \to A \to A$ induces a map $\mathsf{rec}(f_0, f_s) \in \mathsf{nat} \to A$. The computational principle states that recursively defined maps obey the following equalities:

$$\mathsf{rec}(f_0, f_s)\ 0 = f_0$$

$$\mathsf{rec}(f_0, f_s)\ (sx) = f_s\ x\ (\mathsf{rec}(f_0, f_s)\ x).$$

The computational principle implicitly relies on the *no confusion* principle [MG85], which in the case of natural numbers states that every element of nat is built from 0 and s in a unique way. The no confusion principle, which is related to the notion of *deterministic rule set* of [Acz77], is crucial to the evaluation of recursively defined functions.

Technically, note that the higher-order rewriting system obtained by orienting the above equations is constructor based and has no critical pair; see [KOR93] for background in higher-order rewriting.

3.2. Parametric Single-Sorted Signatures

Whereas determinism is inherent to single-sorted, non-parameterized signatures, parameterized signatures are intrinsically non-deterministic in the sense that there is more than one way to derive that a term has a given sort. As a result, not all such signatures support a meaningful theory of recursive definitions.

To rephrase (in a slightly different form) the example of the Introduction, consider the well-known parametric signature Sum of disjoint unions of Table 4. The definitional principle for Sum states that every pair of maps $f_1 \in P_1 \to A$ and $f_2 \in P_2 \to A$ induces a map $f_1 + f_2 \in (\mathsf{Sum}\ P_1\ P_2) \to A$. The computational principle for Sum is given by the equalities

$$(f_1 + f_2)(\mathsf{inj}_1\ x) = f_1(x)$$

$$(f_1 + f_2)(\mathsf{inj}_2\ x) = f_2(x).$$

However, it is not possible any longer to give a computational principle for the parametric signature Union of unions in Table 4 as we overload the function symbol inj. By analogy with Sum, the definitional principle for Union should state that every pair of maps $f_1 \in P_1 \to A$ and $f_2 \in P_2 \to A$ induces a map $f_1 \cup f_2 \in (\mathsf{Union}\ P_1\ P_2) \to A$. The computational principle for Union should be given by the equalities

$$(f_1 \cup f_2)(\mathsf{inj}\ x) = f_1(x)$$

$$(f_1 \cup f_2)(\mathsf{inj}\ x) = f_2(x).$$

The above equalities are inconsistent with equational reasoning as they imply (by symmetry and transitivity of equality) $f_1(x) = f_2(x)$ for arbitrary $f_1 \in P_1 \to A$, $f_2 \in P_2 \to A$, and $x \in P_1 \cap P_2$. In fact, the above inconsistency is due to non-determinism: for $x \in P_1 \cap P_2$, there are two different ways to derive

$$|\vdash_{\Sigma}^{[p_1 \mapsto P_1,\ p_2 \mapsto P_2]} \mathsf{inj}\ x : \mathsf{Union}.$$

TABLE 4

Sums and Unions

	Sum type		Union type
Parameters:	p_1, p_2	Parameters:	p_1, p_2
Sorts:	Sum	Sorts:	Union
Subsort relation:		Subsort relation:	
Declarations:	$\mathsf{inj}_1 : p_1 \to \mathsf{Sum}$	Declarations:	$\mathsf{inj} : p_1 \to \mathsf{Union}$
	$\mathsf{inj}_2 : p_2 \to \mathsf{Sum}$		$\mathsf{inj} : p_2 \to \mathsf{Union}$

TABLE 5

Another Signature of Unions

Union via subtyping	
Parameters:	p_1, p_2
Sorts:	Union'
Subsort relation:	$p_1, p_2 \leqslant \mathsf{Union}'$
Declarations:	

To prevent non-determinism, we require that for every function symbol f and natural number n, there is at most one declaration $f : \boldsymbol{\sigma} \to \bullet$ such that $n = \#\boldsymbol{\sigma}$ (we use $\bullet$ to denote the unique sort of the signature). This condition rules out ill-behaved signatures such as Union and ensures a well-behaved theory of recursive definitions. Indeed, every signature that complies with the above condition meets two fundamental requirements:

1. for every $I \in \mathscr{P} \to \mathbf{Set}$, there is exactly one derivation of $\|\!\vdash^I_\Sigma t : \bullet$ (we implicitly assume that the rule (*Sub*) is not used in the derivations because it is vacuous);

2. the derivation is totally determined by the head function symbol of t and its number of arguments.

Under this requirements, recursive definitions can be evaluated unambiguously.

Remark. Our definition of signature rules out the possibility of declaring a parameter as a subtype of a sort. This restriction is necessary from the point of view of recursive definitions. For reasons similar to the ones above, it is not possible to achieve a well-behaved theory of recursive definitions for the signature Union' defined in Table 5.

3.3. Many-Sorted Signatures

Of course, the counterexample of Union carries over immediately to non-parametric, many-sorted signatures: we only need to view p_1 and p_2 in Table 4 as sorts instead of parameters. Fortunately, one can adopt the same solution as for parameterized single-sorted signatures. In fact, the solution is also adequate for parameterized many-sorted signatures.

DEFINITION 5. A many-sorted signature $\Sigma = (\mathscr{F}, \mathsf{decl})$ over $(\mathscr{P}, (\Lambda, \leqslant))$ is *strictly overloaded* if for every $f \in \mathscr{F}$, $n \in \mathbb{N}$, and $\tau \in \Lambda$, the set $\mathsf{Dom}^\Sigma_{(\tau, n, f)}$ contains at most one-element.

Every non-overloaded many-sorted signature is strictly overloaded but not conversely. For example, the many-sorted signature of natural numbers in Table 6 is strictly overloaded but is not non-overloaded.

TABLE 6

A Strictly Overloaded Signature

Parameters:	
Sorts:	even odd nat
Subsort relation:	
Declarations:	0 : even
	0 : nat
	s : even $\rightarrow$ odd
	s : odd $\rightarrow$ even
	s : nat $\rightarrow$ nat

Every strictly overloaded many-sorted signature meets two fundamental requirements that generalize those of the previous subsection.

1. for every $I \in \mathscr{P} \rightarrow \mathbf{Set}$, there is exactly one derivation of $|\vdash_{\Sigma}^{I} t : \tau$ (we implicitly assume that the rule (*Sub*) is not used in the derivations because it is vacuous);

2. the derivation is totally determined by τ, the head function symbol of t, and its number of arguments.

Subsequently strictly overloaded many-sorted signatures support recursive definitions. In that case, the definitions simultaneously introduce a family of functions $(f_\tau)_{\tau \in \Lambda}$ with $f_\tau \in \mathscr{T}_{\Sigma}^{I}(\tau) \rightarrow J_\tau$, where $J \in \Lambda \rightarrow \mathbf{Set}$ and $I \in \mathscr{P} \rightarrow \mathbf{Set}$ is some fixed interpretation. We do not include a formal description of recursive definitions for such signatures since they form a special class of strictly overloaded order-sorted signatures. A formal description of recursive definitions for the latter may be found in Subsection 3.4.

3.4. Order-Sorted Signatures

In the order-sorted setting, the (*Sub*) rule introduces a further element of nondeterminism. Thus it becomes harder to specify a good class of signatures that support recursive definitions. One appealing way to circumvent the problem is to reduce order-sorted induction to its many-sorted counterpart. In this subsection, we generalize the notion of the strictly overloaded signature to the order-sorted case and show that every strictly overloaded order-sorted signature can be simulated by a strictly overloaded many-sorted signature. The simulation property is then exploited to support recursive definitions for data types specified by strictly overloaded order-sorted signatures.

Remark. It is also possible to reduce order-sorted induction directly to its single-sorted counterpart. However, the reduction of order-sorted induction to its many-sorted counterpart seems more natural. In any case, one can reduce many-sorted induction to single-sorted induction and combine both steps to reduce order-sorted induction to its single-sorted counterpart.

3.4.1. Strict Overloading

Before giving a formal definition of strict overloading, let us return to Definition 5. The requirement there is that $\mathsf{Dom}^{\Sigma}_{(\tau, n, f)}$ should have at most one element. Obviously, such a requirement is overly strong for order-sorted signatures and is not complied with by the order-sorted signatures of natural numbers and Harrop formulae in Table 2. Those examples suggest that it is in fact more appropriate to define a notion of *canonical declaration*. In view of requirement (2) in the previous subsection, one expects a canonical n-ary τ-declaration $f: \boldsymbol{\sigma} \to \tau'$ to verify for every $I \in \mathscr{P} \to \mathbf{Set}$

$$|\vdash^{I}_{\Sigma} f(t_1, ..., t_n) : \tau \Leftrightarrow |\vdash^{I}_{\Sigma} t_i : \sigma_i \qquad \text{for} \quad i = 1, ..., n.$$

Clearly the above equivalence is complied with iff $\boldsymbol{\sigma}$ is the largest n-ary τ-domain of f. If canonical declarations always exist, then for every $I \in \mathscr{P} \to \mathbf{Set}$, every judgment $\vdash^{I}_{\Sigma} t : \tau$ has at most one canonical derivation, where a derivation is canonical if its last rule is not (*Sub*) and all instances of (*Fun*) use canonical declarations. As we shall see in Lemma 8, this forms an acceptable weakening of requirement (1). More generally, the existence of canonical declarations is sufficient to ensure that an order-sorted signature may be simulated by a strictly overloaded many-sorted signature and hence supports a well-behaved theory of recursive definitions.

DEFINITION 6 (Strictly Overloaded Signature). 1. Σ is *strictly overloaded* if for every $(\tau, n, f) \in \mathscr{R}$, the set $\mathsf{Dom}^{\Sigma}_{(\tau, n, f)}$ has a maximal element.

2. The maximal element of $\mathsf{Dom}^{\Sigma}_{(\tau, n, f)}$, when it exists, is called the *canonical n-ary τ-domain of f* and is denoted by $\mathsf{maxdom}^{\Sigma}_{(\tau, n, f)}$.

Strict overloading isolates a class of order-sorted signatures for which nondeterminism is innocuous. Indeed, all strictly overloaded order-sorted signatures, including those of Table 2, support recursive definitions as they can be simulated by strictly overloaded many-sorted signatures. As alluded to above, the idea is to restrict ourselves to canonical declarations: formally, this provides a method to build a many-sorted signature from an order-sorted one and is the key to the theory of recursive definitions. For the remainder of this subsection, we assume given a strictly overloaded signature $\Sigma = (\mathscr{F}, \mathsf{decl})$ over $(\mathscr{P}, (\Lambda, \leqslant))$.

DEFINITION 7. The signature Σ' over $(\mathscr{P}, (\Lambda, =))$ has $\mathscr{F}$ as its set of function symbols and as declarations $f: \mathsf{maxdom}^{\Sigma}_{(\tau, n, f)} \to \tau$, where $(\tau, n, f) \in \mathscr{R}$.

If we take Σ to be the order-sorted signature of natural numbers of Table 2, then Σ' is the many-sorted and strictly overloaded signature of Table 6.

More generally, Σ' is many-sorted and strictly overloaded. It also simulates Σ as expressed in the following lemma.

LEMMA 8. *For every $I \in \mathscr{P} \to \mathbf{Set}$, $\tau \in \Lambda$ and Σ-term t:*

$$|\vdash^{I}_{\Sigma} t : \tau \Leftrightarrow |\vdash^{I}_{\Sigma'} t : \tau.$$

Proof. Both proofs are by induction on the structure of derivations. For the sake of clarity, we write $f :_{\Sigma} \boldsymbol{\sigma} \to \tau$ or $f :_{\Sigma'} \boldsymbol{\sigma} \to \tau$ to emphasize the origin of the declaration.

$\Rightarrow$ We prove a stronger result, namely

$$|\vdash_{\Sigma} t : \tau \wedge \tau \leqslant \rho \Rightarrow |\vdash_{\Sigma'} t : \rho.$$

The only interesting case is when the last rule applied is (*Fun*). So assume that $f :_{\Sigma} (\sigma_1, ..., \sigma_n) \to \tau$ and that the last rule is

$$\frac{|\vdash_{\Sigma} t_1 : \sigma_1 \quad \cdots \quad |\vdash_{\Sigma} t_n : \sigma_n}{|\vdash_{\Sigma} f(t_1, ..., t_n) : \tau}.$$

Let ρ s.t. $\tau \leqslant \rho$. To show $|\vdash_{\Sigma'} f(t_1, ..., t_n) : \rho$. For $i = 1, ..., n$, we have $\sigma_i \leqslant \mathsf{maxdom}^{\Sigma}_{(\rho, n, f)}[i]$. Hence by the induction hypothesis, we have $|\vdash_{\Sigma'} t_i : \mathsf{maxdom}^{\Sigma}_{(\rho, n, f)}[i]$ for $i = 1, ..., n$. By definition of Σ', $f :_{\Sigma'} \mathsf{maxdom}^{\Sigma}_{(\rho, n, f)} \to \rho$ and hence by (*Fun*) $|\vdash_{\Sigma'} f(t_1, ..., t_n) : \rho$.

$\Leftarrow$ The only interesting case is when the last rule applied is (*Fun*). So assume that $f :_{\Sigma'} (\sigma_1, ..., \sigma_n) \to \tau$ and that the last rule is

$$\frac{|\vdash_{\Sigma'} t_1 : \sigma_1 \quad \cdots \quad |\vdash_{\Sigma'} t_n : \sigma_n}{|\vdash_{\Sigma'} f(t_1, ..., t_n) : \tau}.$$

By the induction hypothesis, $|\vdash_{\Sigma} t_i : \sigma_i$ for $i = 1, ..., n$. Moreover, there exists $\tau' \leqslant \tau$ such that $f :_{\Sigma} (\sigma_1, ..., \sigma_n) \to \tau'$. Hence by (*Fun*), $|\vdash_{\Sigma} f(t_1, ..., t_n) : \tau'$. We conclude by applying (*Sub*). ∎

The simulation property suggests that the class of strictly overloaded order-sorted signatures is not essentially more expressive than the class of strictly overloaded many-sorted signatures. In fact, it is our view that strictly overloaded order-sorted signatures provide some shorthand definitions of strictly overloaded many-sorted signatures. With this view, it is also possible to start from a many-sorted signature and to define $\sigma \leqslant \tau$ if τ has more constructors than σ. To our best knowledge, this approach was first suggested by Coquand [Coq92] but, as noticed by Luo [Luo98], this approach is quite limited in the absence of overloading.

3.4.2. A Scheme for Recursive Definitions

Lemma 8 allows us to reduce the problem of recursive definitions for strictly overloaded order-sorted signatures to that of recursive definitions for strictly overloaded many-sorted signatures. This is the path taken below, where we define order-sorted induction without any reference to subtyping.

Throughout the remaining of this subsection, we assume given two maps that $I \in \mathscr{P} \to \mathbf{Set}$ and $J \in \Lambda \to \mathbf{Set}$. For every $l \in (\Lambda \cup \mathscr{P})^\omega$ define $J^\omega(l)$ as

$$J^\omega([\,]) = [\,]$$
$$J^\omega(a :: l) = \begin{cases} J(a) :: J^\omega(l) & \text{if} \quad a \in \Lambda \\ J^\omega(l) & \text{otherwise.} \end{cases}$$

For every $l \in \mathbf{Set}^\omega$ and $B \in \mathbf{Set}$, define $l \to B$ as

$$[\,] \to B = B$$
$$(a :: l) \to B = a \to l \to B.$$

For every $r = (\sigma, n, f) \in \mathscr{R}$, define $\mathscr{H}_r(I, J)$ as

$$\mathscr{H}_r(I, J) = \mathscr{T}^I_\Sigma(\mathsf{maxdom}^\Sigma_r) \to J^\omega(\mathsf{maxdom}^\Sigma_r) \to J(\sigma).$$

Definitional Principle for Order-Sorted Induction. Assume that $F \in \prod r \in \mathscr{R} \cdot \mathscr{H}_r(I, J)$. For every $\sigma \in \Lambda$, $\mathsf{rec}^J_\sigma[I]\ F \in \mathscr{T}^I_\Sigma(\sigma) \to J_\sigma$.

Note that $\leqslant$ is not mentioned explicitly; instead all the information concerning $\leqslant$ is hidden in the definition of Dom. Note also that we do not require that $J_\sigma \subseteq J_\tau$ whenever $\sigma \leqslant \tau$.

Computational Principle for Order-Sorted Induction. If $|\!\vdash^I f(w_1, ..., w_n) : \sigma$ then

$$\mathsf{rec}^J_\sigma[I]\ F(f(w_1, ..., w_n))$$
$$= F_r\, w_1 \cdots w_n (\mathsf{rec}^J_{\mathsf{maxdom}^\Sigma_r[i_1]}[I]\ F\, w_{i_1}) \cdots (\mathsf{rec}^J_{\mathsf{maxdom}^\Sigma_r[i_k]}[I]\ F\, w_{i_k})$$

where

1. $r = (\sigma, n, f)$,
2. $i_1 < \cdots < i_k$,
3. for $1 \leqslant j \leqslant n$, $\mathsf{maxdom}^\Sigma_r[j] \in \Lambda \Leftrightarrow J \in \{I_1, ..., i_k\}$.

3.4.3. Examples

Lists. This is the parametric signature defined in Table 2. Assume that $I \in \mathscr{P} \to \mathbf{Set}$ and $J \in \Lambda \to \mathbf{Set}$.

By unfolding the definition of $\mathscr{R}$ and $\mathscr{H}$ one obtains (for readability, we write $\mathsf{XList}[I]$ for $\mathscr{T}^I_\Sigma(\mathsf{XList})$, where XList ranges over Λ)

$$\mathscr{R} = \{(\mathsf{List}, 0, \mathsf{nil}), (\mathsf{NeList}, 1, \mathsf{cons}), (\mathsf{List}, 1, \mathsf{cons})\}$$
$$\mathscr{H}_{(\mathsf{List}, 0, \mathsf{nil})}(I, J) = J_{\mathsf{List}}$$
$$\mathscr{H}_{(\mathsf{XList}, 1, \mathsf{cons})} = \mathsf{List}[I] \to I(p) \to J_{\mathsf{List}} \to J_{\mathsf{XList}}.$$

The definitional and computational principles are instantiated to the following: if $F \in \prod r \in \mathscr{R} \cdot \mathscr{H}_r(I, J)$ then $\mathsf{rec}^J_{\mathsf{XList}}[I]\ F \in \mathsf{XList}[I] \to J_{\mathsf{XList}}$ and

$$\mathsf{rec}^J_{\mathsf{List}}[I]\ F\ \mathsf{nil} = F_{(\mathsf{List},\,0,\,\mathsf{nil})}$$

$$\mathsf{rec}^J_{\mathsf{XList}}[I]\ F\ (\mathsf{cons}(a, l)) = F_{(\mathsf{XList},\,1,\,\mathsf{cons})}\ a\ l\ (\mathsf{rec}^J_{\mathsf{List}}[I]\ F\ l).$$

Natural Numbers. This is the signature defined in Table 1. Assume that $J \in \Lambda \to \mathbf{Set}$ (we omit I as the signature is non-parametric).

By unfolding the definition of $\mathscr{R}$ and $\mathscr{H}$ one obtains

$$\mathscr{R} = \{(\mathsf{even}, 0, 0), (\mathsf{nat}, 0, 0), (\mathsf{even}, 1, s), (\mathsf{nat}, 1, s), (\mathsf{odd}, 1, s)\}$$

$$\mathscr{H}_{(\mathsf{even},\,0,\,0)}(J) = J_{\mathsf{even}}$$

$$\mathscr{H}_{(\mathsf{nat},\,0,\,0)}(J) = J_{\mathsf{nat}}$$

$$\mathscr{H}_{(\mathsf{even},\,1,\,s)}(J) = \mathsf{odd} \to J_{\mathsf{odd}} \to J_{\mathsf{even}}$$

$$\mathscr{H}_{(\mathsf{nat},\,1,\,s)}(J) = \mathsf{nat} \to J_{\mathsf{nat}} \to J_{\mathsf{nat}}$$

$$\mathscr{H}_{(\mathsf{odd},\,1,\,s)}(J) = \mathsf{even} \to J_{\mathsf{even}} \to J_{\mathsf{odd}}.$$

The definitional and computational principles are instantiated the following: if $F \in \prod r \in \mathscr{R} \cdot \mathscr{H}_r(J)$ then $\mathsf{rec}^J_\tau F \in \tau \to J_\tau$ for $\tau \in \{\mathsf{even}, \mathsf{odd}, \mathsf{nat}\}$ and

$$\mathsf{rec}^J_{\mathsf{even}}\ F\ 0 = F_{(\mathsf{even},\,0,\,0)}$$

$$\mathsf{rec}^J_{\mathsf{nat}}\ F\ 0 = F_{(\mathsf{nat},\,0,\,0)}$$

$$\mathsf{rec}^J_{\mathsf{even}}\ F\ (sx) = F_{(\mathsf{even},\,1,\,s)}\ x\ (\mathsf{rec}^J_{\mathsf{odd}}\ F\ x)$$

$$\mathsf{rec}^J_{\mathsf{odd}}\ F\ (sx) = F_{(\mathsf{odd},\,1,\,s)}\ x\ (\mathsf{rec}^J_{\mathsf{even}}\ F\ x)$$

$$\mathsf{rec}^J_{\mathsf{nat}}\ F\ (sx) = F_{(\mathsf{nat},\,1,\,s)}\ x\ (\mathsf{rec}^J_{\mathsf{nat}}\ F\ x).$$

Of course, one can recover from the above principles the usual recursion principle for natural numbers. Assume that we are given a set B together with $f_0 \in B$ and $f_s \in \mathsf{nat} \to B \to B$. To recover the standard principles, set $J_\tau = B$ for $\tau \in \{\mathsf{even}, \mathsf{odd}, \mathsf{nat}\}$ and take F to be such that

$$F_{(\mathsf{even},\,0,\,0)} = f_0$$

$$F_{(\mathsf{nat},\,0,\,0)} = f_0$$

$$F_{(\mathsf{even},\,1,\,s)} = \lambda x \in \mathsf{even} \cdot f_s\ x$$

$$F_{(\mathsf{odd},\,1,\,s)} = \lambda x \in \mathsf{odd} \cdot f_s\ x$$

$$F_{(\mathsf{nat},\,1,\,s)} = f_s.$$

4. THE SYSTEM $IF^{\omega}_{\leqslant}$

In this section, we introduce the system $IF^{\omega}_{\leqslant}$. For the sake of clarity, we focus on a system with a single order-sorted inductive type generated by a fixed strictly overloaded signature $\Sigma = (\mathscr{F}, \mathsf{decl})$ over $(\mathscr{P}, (\Lambda, \leqslant))$; a more general approach would consist in providing a scheme for introducing order-sorted inductive types.

DEFINITION 9 (Expressions). Let $\mathbb{V}^*$ and $\mathbb{V}^{\square}$ be disjoint countably infinite sets of constructor variables and of object variables, respectively. The sets $\mathbb{K}$ of *kinds*, $\mathbb{C}$ of *constructors*, and $\mathbb{O}$ of *objects* are defined by the abstract syntaxes

$$\mathbb{K} = * \mid \mathbb{K} \to \mathbb{K}$$

$$\mathbb{C} = \mathbb{V}^{\square} \mid \top^{\mathbb{K}} \mid \mathbb{C} \to \mathbb{C} \mid \Lambda \mid \Pi \mathbb{V}^{\square} \leqslant \mathbb{C} : \mathbb{K} \cdot \mathbb{C} \mid \lambda \mathbb{V}^{\square} : \mathbb{K} \cdot \mathbb{C} \mid \mathbb{C}\ \mathbb{C}$$

$$\mathbb{O} = \mathbb{V}^* \mid \lambda \mathbb{V}^* : \mathbb{C} \cdot \mathbb{O} \mid \mathbb{O}\ \mathbb{O} \mid \lambda \mathbb{V}^{\square} \leqslant \mathbb{C} : \mathbb{K} \cdot \mathbb{O} \mid \mathbb{O}\ \mathbb{C} \mid \mathscr{F} \mid \mathsf{rec}^{J}_{\Lambda}[I]\ F,$$

where F, I, and J range over $\mathscr{R} \to \mathbb{O}$, $\mathscr{P} \to \mathbb{C}$, and $\Lambda \to \mathbb{C}$, respectively.

Free and bound variables are defined as usual. Moreover, we let $\cdot[\cdot/\cdot]$ denote the standard substitution operator. In addition, we let $\mathsf{size}(\mathscr{P})$ denote the size of $\mathscr{P}$ and assume that $p_1, \ldots, p_{\mathsf{size}(\mathscr{P})}$ is a fixed enumeration of $\mathscr{P}$. Finally, we use specific identifiers to range over given categories of expressions,

Identifier	Range over
$x, y, \ldots$	$\mathbb{V}^*$
$\alpha, \beta, \ldots$	$\mathbb{V}^{\square}$
K, K'	$\mathbb{K}$
$A, A', A_i, B, B_i, C, \ldots$	$\mathbb{C}$
$t, u, t_i, w_i, \ldots$	$\mathbb{O}$
$M, N, P, Q, \ldots$	$\mathbb{K} \cup \mathbb{C} \cup \mathbb{O}$
$I, I', \ldots$	$\mathscr{P} \to \mathbb{C}$
$J, J', \ldots$	$\Lambda \to \mathbb{C}$
$\sigma, \tau, \rho, \ldots$	Λ
$p, p', \ldots$	$\mathscr{P}$

and use the following abbreviations (note that the compounds on the left-hand side are not expressions of the system):

Compound	Abbreviates
$\alpha : K$	$\alpha \leqslant \top^{K} : K$
$\sigma\ I$	$\sigma\ I(p_1) \cdots I(p_{\mathsf{size}(\mathscr{P})})$
$p\ I$	$I(p)$

The computational behavior of $IF^{\omega}_{\leqslant}$ is given by the notion of β-reduction that stems from λ-calculus and by the notion of ι-reduction that stems from inductive types.

DEFINITION 10 (Reduction Calculus). 1. *β_ω-reduction* $\to_{\beta_\omega} \subseteq \mathbb{C} \times \mathbb{C}$ is defined as the compatible closure of

$$(\lambda\alpha : K \cdot A)\ B \to_{\beta_\omega} A[B/\alpha].$$

2. *β_0-reduction* $\to_{\beta_0}$ is defined as the compatible closure of

$$(\lambda x : A \cdot t)\ u \to_{\beta_0} t[u/x].$$

3. *β_2-reduction* $\to_{\beta_2}$ is defined as the compatible closure of

$$(\lambda\alpha \leqslant C : K \cdot t)\ B \to_{\beta_2} t[B/\alpha].$$

4. *ι-reduction* $\to_\iota$ is defined as the compatible closure of

$$\mathsf{rec}^J_\sigma[I]\ F\ (f\ \mathbf{A}\ w_1 \cdots w_n)$$
$$\to_\iota F_r\ w_1 \cdots w_n (\mathsf{rec}^J_{\mathsf{maxdom}^\Sigma_r[i_1]}\ [I]\ F\ w_{i_1}) \cdots (\mathsf{rec}^J_{\mathsf{maxdom}^\Sigma_r[i_k]}\ [I]\ F\ w_{i_k}),$$

where

(a) $r = (\sigma, n, f)$,

(b) $i_1 < \cdots < i_k$,

(c) for $1 \leqslant j \leqslant n$, $\mathsf{maxdom}^\Sigma_r[j] \in \Lambda \leftrightarrow j \in \{i_1, \ldots, i_k\}$

5. *β-reduction* $\to_\beta$ is defined as $\to_{\beta_\omega} \cup \to_{\beta_0} \cup \to_{\beta_2}$.

6. *$\beta\iota$-reduction* $\to_{\beta\iota}$ is defined as $\to_\beta \cup \to_\iota$.

Note that, in the contraction rule for ι-reduction, we do not impose any relationship between I and $\mathbf{A}$. In the following, we let $=_{\beta_\omega}$ denote the reflexive-symmetric-transitive closure of $\to_{\beta_\omega}$.

DEFINITION 11 (Typing Rules). 1. The set $\mathbb{G}$ of context is defined by the abstract syntax

$$\mathbb{G} = \cdot \mid \mathbb{G}, \mathbb{V}^* : \mathbb{C} \mid \mathbb{G}, \mathbb{V}^\square \leqslant \mathbb{C} : \mathbb{K}.$$

We let Γ, Γ', Δ, ... range over contexts.

2. Judgments are triples of one of the following forms:

$$\Gamma \vdash A : K \qquad \textit{kinding judgment}$$
$$\Gamma \vdash t : A \qquad \textit{typing judgment}$$
$$\Gamma \vdash A \leqslant B \qquad \textit{subtyping judgment}.$$

3. The relations $\Gamma \vdash A : K$, $\Gamma \vdash t : A$, and $\Gamma \vdash A \leqslant B$ are defined by the rules of Tables 7 and 8, where $\Gamma \vdash A \leqslant {}^*B$ is used as an abbreviation for

$$\Gamma \vdash A \leqslant B \wedge \Gamma \vdash A : * \wedge \Gamma \vdash B : *.$$

4. If $\vdash t : A$ is derivable, then Γ, t, and A are *legal*. Similarly, if $\Gamma \vdash A : K$ is derivable, then Γ and A are *legal*.

TABLE 7

Rules for Derivations

(Start)	$\dfrac{\Gamma \vdash A : *}{\Gamma, x : A \vdash x : A}$	if $x \notin \Gamma$ and $x \in \mathbb{V}^*$
(Bounded start)	$\dfrac{\Gamma \vdash A : K}{\Gamma, \alpha \leqslant A : K \vdash \alpha : K}$	if $\alpha \notin \Gamma$ and $\alpha \in \mathbb{V}^{\square}$
(Weakening)	$\dfrac{\Gamma \vdash M : N \quad \Gamma \vdash A : *}{\Gamma, x : A \vdash M : N}$	if $x \notin \Gamma$
(Bounded weakening)	$\dfrac{\Gamma \vdash M : N \quad \Gamma \vdash A : K}{\Gamma, \alpha \leqslant A : K \vdash M : N}$	if $\alpha \notin \Gamma$
(Top)	$\dfrac{}{\vdash \top^K : K}$	
(Product)	$\dfrac{\Gamma \vdash A : * \quad \Gamma \vdash B : *}{\Gamma \vdash A \to B : *}$	
(Bounded product)	$\dfrac{\Gamma, \alpha \leqslant A : K \vdash B : *}{\Gamma \vdash \Pi\alpha \leqslant A : K.B : *}$	
(Application)	$\dfrac{\Gamma \vdash M : P \to Q \quad \Gamma \vdash N : P}{\Gamma \vdash MN : Q}$	
(Bounded application)	$\dfrac{\Gamma \vdash t : \Pi\alpha \leqslant A : K.B \quad \Gamma \vdash A' \leqslant A}{\Gamma \vdash tA' : B[A'/\alpha]}$	
(Object abstraction)	$\dfrac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.t : A \to B}$	
(Constructor abstraction)	$\dfrac{\Gamma, \alpha : K_1 \vdash B : K_2}{\Gamma \vdash \lambda\alpha : K.B : K_1 \to K_2}$	
(Bounded abstraction)	$\dfrac{\Gamma, \alpha \leqslant A : K \vdash t : B}{\Gamma \vdash \lambda\alpha \leqslant A : K.t : \Pi\alpha \leqslant A : K.B}$	
(Sort)	$\dfrac{\Gamma \vdash A_i : * \text{ for } 1 \leqslant i \leqslant \mathsf{size}(\mathscr{P})}{\Gamma \vdash \sigma\, A_1 \cdots A_{\mathsf{size}(\mathscr{P})} : *}$	if $\sigma \in \Lambda$
(Function)	$\dfrac{\Gamma \vdash \sigma\, \mathbf{A} : * \quad \Gamma \vdash t_i : \tau_i\, \mathbf{A} \quad \text{for } 1 \leqslant i \leqslant n}{\Gamma \vdash h\, \mathbf{A} t_1 \cdots t_n : \sigma\, \mathbf{A}}$	if $h : (\tau_1, ..., \tau_n) \to \sigma$
(Recursion)	$\dfrac{\Gamma \vdash F_r : \mathscr{H}_r(I, J) \quad \forall r \in \mathscr{R}}{\Gamma \vdash \mathsf{rec}^J_\sigma[I]\, F : (\sigma\, I) \to J_\sigma}$	if $\sigma \in \Lambda$
(Subsumption)	$\dfrac{\Gamma \vdash u : A \quad \Gamma \vdash A \leqslant B}{\Gamma \vdash u : B}$	

TABLE 8

Rules for Subtyping

($\leqslant$-sort)	$\dfrac{\Gamma \vdash \sigma\ \mathbf{A} : *}{\Gamma \vdash \sigma\ \mathbf{A} \leqslant \sigma'\ \mathbf{A}}$	if $\sigma \leqslant \sigma'$
($\leqslant$-start)	$\dfrac{\Gamma \vdash A : K}{\Gamma, \alpha \leqslant A : K \vdash \alpha \leqslant A}$	if $\alpha \notin \Gamma$
($\leqslant$-weakening)	$\dfrac{\Gamma \vdash A \leqslant A' \quad \Gamma \vdash B : *}{\Gamma, x : B \vdash A \leqslant A'}$	if $x \notin \Gamma$
($\leqslant$-bounded weakening)	$\dfrac{\Gamma \vdash A \leqslant A' \quad \Gamma \vdash B : K}{\Gamma, \alpha \leqslant B : K \vdash A \leqslant A'}$	if $\alpha \notin \Gamma$
($\leqslant$-top)	$\dfrac{\Gamma \vdash A : K}{\Gamma \vdash A \leqslant \top^K}$	
($\leqslant$-product)	$\dfrac{\Gamma \vdash A_1' \leqslant^* A_1 \quad \Gamma \vdash A_2 \leqslant^* A_2'}{\Gamma \vdash A_1 \to A_2 \leqslant A_1' \to A_2'}$	
($\leqslant$-bounded product)	$\dfrac{\Gamma, \alpha \leqslant B : K \vdash A \leqslant^* A'}{\Gamma \vdash (\Pi\alpha \leqslant B : K.A) \leqslant (\Pi\alpha \leqslant B : K.A')}$	
($\leqslant$-application)	$\dfrac{\Gamma \vdash A \leqslant A' \quad \Gamma \vdash AB : K}{\Gamma \vdash AB \leqslant A'B}$	
($\leqslant$-abstraction)	$\dfrac{\Gamma, \alpha : K \vdash A \leqslant A'}{\Gamma \vdash \lambda\alpha : K.A \leqslant \lambda\alpha : K.A'}$	
($\leqslant$-transitivity)	$\dfrac{\Gamma \vdash A \leqslant A' \quad \Gamma \vdash A' \leqslant B}{\Gamma \vdash A \leqslant B}$	
($\leqslant$-conversion)	$\dfrac{\Gamma \vdash A : K \quad \Gamma \vdash A' : K}{\Gamma \vdash A \leqslant A'}$	if $A =_{\beta_\omega} A'$

The typing and subtyping rules of $IF^{\omega}_{\leqslant}$ are those of $F^{\omega}_{\leqslant}$ together with new rules to handle order-sorted data types and recursive definitions. Some points are worth noting:

1. sorts are required to be monotonic so that, e.g., List even $\leqslant$ List nat is derivable from even $\leqslant$ nat;

2. function symbols are parameterized so a legal cons-expression will be of the form cons $A\ a\ l$, where A is a legal type, $a : A$, and $l :$ List A;

3. in the (recursion) rule, it is implicitly assumed that for every $\tau \in \Lambda$, $\tau\ I$ and J_τ are legal types in Γ.

5. PROPERTIES OF $IF^{\omega}_{\leqslant}$

This section establishes some fundamental properties of $IF^{\omega}_{\leqslant}$, including confluence and strong normalization of the reduction calculus, decidability of type checking, and subject reduction.

5.1. Confluence and Strong Normalization

Two important properties of a typed λ-calculus are confluence and strong normalization. From an operational point of view, confluence establishes that all reduction strategies yield the same result whereas strong normalization establishes that all reduction sequences terminate.

PROPOSITION 12 (Confluence). *$\rightarrow_{\beta\iota}$ is confluent.*

Proof. By the standard technique of Tait and Martin-Löf. ▮

Strong normalization for constructors is easy to establish.

PROPOSITION 13 (Constructor Strong Normalization). *Assume that $\Gamma \vdash A : K$. Then A is β-strongly normalizing; i.e., all $\rightarrow_\beta$-reduction sequences starting from A terminate.*

Proof. The kinding fragment of $IF^\omega_\leqslant$ is equivalent to a simply typed λ-calculus with constants. ▮

Strong normalization for objects is derived from a model construction based on saturated sets.

THEOREM 14 (Object Strong Normalization). *Assume that $\Gamma \vdash t : A$. Then t is $\beta\iota$-strongly normalizing; i.e., all $\rightarrow_{\beta\iota}$-reduction sequences starting from t terminate.*

Proof. The result follows from Corollary 30 below. ▮

5.2. The Model Construction

This subsection is devoted to a model construction for $IF^\omega_\leqslant$. Our construction is an adaptation of the PER models for $F^\omega_\leqslant$; see, e.g., [Com95]. The main difference with [Com95] is that PERs are replaced with saturated sets. Of course, the model is also suitably extended so as to accommodate inductive types. Because of the similarity with other model constructions, see, e.g. [Com95], most details are omitted. Basic properties of the system, which parallel those developed in [Com95], are used fairly intensively.

5.2.1. Saturated Sets

First, we define saturated sets and state some of their closure properties. For convenience, we prefer to define saturated sets as sets of erased objects rather than objects.

DEFINITION 15. 1. The set $\mathbb{E}$ of *erased objects* is defined by the abstract syntax

$$\mathbb{E} = \mathbb{V}^* \mid \lambda\, \mathbb{V}^* \cdot \mathbb{E} \mid \mathbb{E}\ \mathbb{E} \mid \mathscr{F} \mid \mathsf{rec}_A\, H,$$

where H ranges over $\mathscr{R} \to \mathbb{E}$.

2. *Erased β-reduction* $\rightarrow_{\underline{\beta}}$ is defined as the compatible closure of

$$(\lambda x \cdot M)\, N \rightarrow_{\underline{\beta}} M[N/x].$$

3. *Erased ι-reduction* $\to_{\underline{\iota}}$ is defined as the compatible closure of

$$\mathsf{rec}_\sigma\, H\,(f\, w_1 \cdots w_n) \to_{\underline{\iota}} H_r\, w_1 \cdots w_n\, (\mathsf{rec}_{\mathsf{maxdom}_r^\Sigma[i_1]}\, H\, w_{i_1}) \cdots (\mathsf{rec}_{\mathsf{maxdom}_r^\Sigma[i_k]}\, H\, w_{i_k}),$$

where

(a) $r = (\sigma, n, f)$,

(b) $i_1 < \cdots < i_k$,

(c) for $1 \leqslant j \leqslant n$, $\mathsf{maxdom}_r^\Sigma[j] \in \Lambda \Leftrightarrow j \in \{i_1, \ldots, i_k\}$.

4. The set of *$\underline{\beta\iota}$-strongly normalizing erased objects* is denoted by SN.

5. The *erasure map* $|\cdot| : \mathbb{O} \to \mathbb{E}$ is defined by induction on the structure of objects as follows:

$$\begin{aligned} |x| &= x \\ |\lambda x : A \cdot t| &= \lambda x \cdot |t| \\ |t\, u| &= |t|\, |u| \\ |\lambda\alpha \leqslant A : K \cdot t| &= |t| \\ |t\, A| &= |t| \\ |f| &= f \\ |\mathsf{rec}_\tau^J[I]\, F| &= \mathsf{rec}_\tau\, |F|, \end{aligned}$$

where $|\cdot|$ is extended to $\mathscr{R} \to \mathbb{O}$ in the obvious way.

The usefulness of erased objects is captured in the following result.

LEMMA 16. *If $\Gamma \vdash t : A$ and $|t| \in \mathsf{SN}$ then t is $\beta\iota$-strongly normalizing.*

Proof. Prove by induction on the structure of objects:

$$\begin{aligned} t \to_{\beta_0} u &\Rightarrow |t| \to_{\underline{\beta}} |u| \\ t \to_{\beta_2} u &\Rightarrow |t| = |u| \\ t \to_{\beta_\omega} u &\Rightarrow |t| = |u| \\ t \to_\iota u &\Rightarrow |t| \to_{\underline{\iota}} |u|. \end{aligned}$$

Now assume that t is not $\beta\iota$-strongly normalizing, i.e., admits an infinite $\beta\iota$-reduction sequence. Legal constructors are strongly normalizing by Proposition 13, hence the reduction sequence must contain infinitely many $\to_{\beta_0\iota}$-reduction steps. Hence there is an infinite $\underline{\beta\iota}$-reduction sequence starting from $|t|$. ∎

We now turn to the definition of saturated sets. We start by introducing specific subset of SN.

DEFINITION 17. The set BA of *base terms* is defined inductively as follows:

1. $\mathbb{V}^* \subseteq \mathsf{BA}$;
2. if $M \in \mathsf{BA}$ and $N \in \mathsf{SN}$, then $M\,N \in \mathsf{BA}$;
3. if $M \in \mathsf{BA}$, $\tau \in \Lambda$, and $H \in \mathscr{R} \to \mathsf{SN}$, then $\mathsf{rec}_\tau\, H\, M \in \mathsf{BA}$.

The following definition is required in order to formulate the notion of the saturated set.

DEFINITION 18. 1. The *key-redex* of $M \in \mathbb{E}$ is defined inductively as follows:

(a) if M is a $\underline{\beta\iota}$-redex, then M is its own key-redex;

(b) if M has key-redex N, then $M\,P$ has key-redex N;

(c) if M has key-redex N, then $\mathsf{rec}_\tau\, H\, M$ has key-redex N.

2. *Key-reduction* $\to_k$ is the smallest relation on erased objects such that $M \to_k N$ if N is obtained from M by contracting its (unique, if it exists) key-redex.

Saturated sets are defined as subsets of SN with suitable closure properties.

DEFINITION 19. A set X of erased objects is *saturated* if

1. $\mathsf{BA} \subseteq X \subseteq \mathsf{SN}$
2. $\forall M \in \mathbb{E}.\ M \in \mathsf{SN} \wedge \mathbf{kred}(M) \in X \Rightarrow M \in X$.

The collection of saturated sets is denoted by **SAT**.

Saturated sets enjoy some standard closure properties: for example, $\mathsf{SN} \in \mathbf{SAT}$. Moreover, **SAT** is closed under arbitrary non-empty intersections. Finally, if $X, Y \in \mathbf{SAT}$, then $X \to Y \in \mathbf{SAT}$, where

$$X \to Y = \{M \in \mathsf{SN} \mid \forall N \in X \cdot M\,N \in Y\}.$$

5.2.2. Interpretation of Kinds

Kinds are interpreted as elements of the full type structure over **SAT**.

DEFINITION 20. 1. The set Ω is the smallest set containing **SAT** and closed under function space formation.

2. The interpretation $\langle\!\langle \cdot \rangle\!\rangle : \mathsf{Kind} \to \Omega$ is defined inductively as follows:

$$\langle\!\langle * \rangle\!\rangle = \mathbf{SAT}$$
$$\langle\!\langle K_1 \to K_2 \rangle\!\rangle = \langle\!\langle K_2 \rangle\!\rangle \to \langle\!\langle K_2 \rangle\!\rangle.$$

Every element of Ω can be endorsed with the structure of a partial order with a top element.

DEFINITION 21. 1. The relation $\lhd_K$ is defined by induction on the structure of $K \in \mathbb{K}$:

(a) $f \lhd_* g$ if $f \subseteq g$,

(b) $f \lhd_{K_1 \to K_2} g$ if $\forall x \in \langle\!\langle K_1 \rangle\!\rangle \cdot f(x) \lhd_{K_2} g(x)$.

2. The element $c^K \in \langle\!\langle K \rangle\!\rangle$ is defined by induction on the structure of $K \in \mathbb{K}$:

$$c^* = \mathsf{SN}$$

$$c^{K_1 \to K_2} = \boldsymbol{\lambda} f \in \langle\!\langle K_1 \rangle\!\rangle \cdot c^{K_2}$$

It is readily checked that, for every $K \in \mathbb{K}$, $\lhd_K$ is a partial order over $\langle\!\langle K \rangle\!\rangle$ with c^K as its top element.

5.2.3. *Interpretation of Constructors*

Constructors are interpreted as elements of elements of Ω. More precisely, we define a constructor interpretation $[\![\cdot]\!]_\xi : \mathbb{C} \to \bigcup \Omega$, where ξ is a constructor valuation. We then show that, under suitable conditions, a valuation ξ satisfies

$$\Gamma \vdash A : K \Rightarrow [\![A]\!]_\xi \in \langle\!\langle K \rangle\!\rangle.$$

In particular, types are interpreted as saturated sets since $\langle\!\langle * \rangle\!\rangle = \mathbf{SAT}$. According to the above definition, sorts are interpreted as functions between saturated sets. The next definition conveys such an interpretation.

DEFINITION 22. Let $\zeta : \mathscr{P} \to \mathbf{SAT}$. Define $[\![\cdot]\!]_\zeta : (\Lambda \cup \mathscr{P}) \to \mathbf{SAT}$ as follows:

1. if $p \in \mathscr{P}$, then $[\![p]\!]_\zeta = \zeta(p)$;
2. if $\tau \in \Lambda$, then $[\![\tau]\!]_\zeta$ is the saturated set defined by the clauses
   (a) $\mathsf{BA} \subseteq [\![\tau]\!]_\zeta$,
   (b) if $f : (\sigma_1, ..., \sigma_n) \to \tau$ and $t_i \in [\![\sigma_i]\!]_\zeta$ for $i = 1, ..., n$ then $f\, t_1 \cdots t_n \in [\![\tau]\!]_\zeta$;
   (c) if $M \in \mathsf{SN}$ and $\mathbf{kred}(M) \in [\![\tau]\!]_\zeta$, then $M \in [\![\tau]\!]_\zeta$.

The interpretation of sorts is monotonic; i.e., $[\![\sigma]\!]_\zeta \subseteq [\![\sigma]\!]_{\zeta'}$ if $\zeta(p) \subseteq \zeta'(p)$ for every $p \in \mathscr{P}$. It also preserves the ordering between sorts', i.e., $[\![\sigma]\!]_\zeta \subseteq [\![\sigma']\!]_\zeta$ if $\sigma \leqslant \sigma'$.

DEFINITION 23. 1. A *constructor valuation* is a map $\mathbb{V}^\square \to \bigcup \Omega$.

2. For every constructor valuation ξ, $M \in \bigcup \Omega$, and $\alpha \in \mathbb{V}^\square$, the constructor valuation $\xi(\alpha := M)$ is defined as the unique valuation ξ' such that $\xi'(\alpha) = M$ and $\xi'(y) = \xi(y)$ if $\alpha \neq y$.

3. For every constructor valuation ξ, the interpretation $[\![\cdot]\!]_\xi : \mathbb{C} \to \bigcup \Omega$ is defined as

$$[\![\alpha]\!]_\xi = \xi(\alpha)$$

$$[\![\top^K]\!]_\xi = c^K$$

$$[\![A \to B]\!]_\xi = [\![A]\!]_\xi \to [\![B]\!]_\xi$$

$$[\![\Pi\alpha \leqslant A : K \cdot B]\!]_\xi = \bigcap_{z \lhd_K [\![A]\!]_\xi} [\![B]\!]_{\xi(\alpha := z)}$$

$$[\![\lambda\alpha : K \cdot A]\!]_\xi = \lambda z \in \langle\!\langle K \rangle\!\rangle . [\![A]\!]_{\xi(\alpha := z)}$$

$$[\![AB]\!]_\xi = [\![A]\!]_\xi ([\![B]\!]_\xi)$$

$$[\![\tau\ A_1 \cdots A_{\mathsf{size}(\mathscr{P})}]\!]_\xi = [\![\tau]\!]_{[p_1 \mapsto [\![A_1]\!]_\xi, \ldots, p_{\mathsf{size}(\mathscr{P})} \mapsto [\![A_{\mathsf{size}(\mathscr{P})}]\!]_\xi]}.$$

The interpretation is well behaved with respect to substitution and reduction.

LEMMA 24. *Let ξ be a constructor valuation.*

1. *If $\alpha \in \mathbb{V}^\square$ and $M, P \in \mathbb{C}$, then $[\![M]\!]_{\xi(\alpha := [\![P]\!]_\xi)} = [\![M[P/\alpha]]\!]_\xi$.*
2. *If $M \in \mathbb{C}$ and $M \to_\beta N$, then $[\![M]\!]_\xi = [\![N]\!]_\xi$.*

Next, we define the notion of satisfaction.

DEFINITION 25. 1. Let ξ be a constructor valuation.

(a) ξ *satisfies a context* Γ, written $\xi \models \Gamma$, if for every $(\alpha \leqslant A : K) \in \Gamma$, $\xi(\alpha) \in \langle\!\langle K \rangle\!\rangle$ and $\xi(\alpha) \lhd_K [\![A]\!]_\xi$.

(b) ξ *satisfies a kinding judgment* $\Gamma \vdash A : K$, written $\Gamma \models_\xi A : K$, if $[\![A]\!]_\xi \in \langle\!\langle K \rangle\!\rangle$.

(c) ξ *satisfies a subtyping judgment* $\Gamma \vdash A \leqslant B$, written $\Gamma \models_\xi A \leqslant B$, if $[\![A]\!]_\xi \lhd_K [\![B]\!]_\xi$, where $K \in \mathbb{K}$ is the unique kind s.t. $\Gamma \vdash A : K$ is derivable.

2. A kinding judgment $\Gamma \vdash A : K$ is *sound*, written $\Gamma \models A : K$, if for every constructor valuation ξ,

$$\xi \models \Gamma \Rightarrow \Gamma \models_\xi A : K.$$

3. A subtyping judgment $\Gamma \vdash A \leqslant B$ is *sound*, written $\Gamma \models A \leqslant B$, if for every constructor valuation ξ;

$$\xi \models \Gamma \Rightarrow \Gamma \models_\xi A \leqslant B.$$

We have:

PROPOSITION 26 (Soundness for Kinding and Subtyping Judgments).

1. $\Gamma \vdash A : K \Rightarrow \Gamma \models A : K$.
2. $\Gamma \vdash A \leqslant B \Rightarrow \Gamma \models A \leqslant B$.

Proof. By induction on the structure of the derivations. ▮

5.2.4. Interpretation of Objects

We now interpret objects by defining an object interpretation $(\!|\cdot|\!)_\rho : \mathbb{O} \to \mathbb{E}$, where ρ is an object valuation. We then show that under suitable conditions valuations verify

$$\Gamma \vdash t : A : * \Rightarrow (\!|t|\!)_\rho \in [\![A]\!]_\xi.$$

By interpreting types as saturated sets and taking $\rho = |\cdot|$, we conclude that

$$\Gamma \vdash t : A : * \Rightarrow |t| \in [\![A]\!]_\xi \in \mathbf{Sat}.$$

Now $[\![A]\!]_\xi \subseteq \mathsf{SN}$ so we can conclude that $|t| \in \mathsf{SN}$ and hence t is $\beta\iota$-strongly normalizing.

DEFINITION 27. 1. An *object valuation* is a map $\rho: \mathbb{V}^* \to \mathbb{E}$.

2. For every object valuation $\rho, e \in \mathbb{E}$, and $x \in \mathbb{V}^*$, the object valuation $\rho(x := e)$ is defined as the unique valuation ρ' such that $\rho'(x) = e$ and $\rho'(y) = \rho(y)$ if $x \neq y$.

3. For every object valuation ρ, the map $(\!|\cdot|\!)_\rho: \mathbb{O} \to E$ is defined inductively as

$$\begin{aligned}
(\!|x|\!)_\rho &= \rho x \\
(\!|\lambda x : A \cdot t|\!)_\rho &= \lambda x \cdot (\!|t|\!)_{\rho(x := x)} \\
(\!|t\ u|\!)_\rho &= (\!|t|\!)_\rho\ (\!|u|\!)_\rho \\
(\!|\lambda \alpha \leqslant A : K \cdot t|\!)_\rho &= (\!|t|\!)_\rho \\
(\!|t\ A|\!)_\rho &= (\!|t|\!)_\rho \\
(\!|f|\!)_\rho &= f \\
(\!|\mathsf{rec}^J_\tau[I]\ F|\!)_\rho &= \mathsf{rec}_\tau (\!|F|\!)_\rho,
\end{aligned}$$

where $(\!|\cdot|\!)_\rho$ is extended to $\mathscr{R} \to \mathbb{O}$ in the obvious way.

4. A *valuation* is a pair (ξ, ρ), where ξ is a constructor valuation and ρ is an object valuation.

Satisfaction is defined in essentially the same way as for constructors.

DEFINITION 28. 1. (ξ, ρ) be a valuation.

(a) (ξ, ρ) *satisfies a context* Γ, written $(\xi, \rho) \models \Gamma$, if $\xi \models \Gamma$ and for every $x : A \in \Gamma$, $\rho(x) \in [\![A]\!]_\xi$.

(b) (ξ, ρ) *satisfies a typing judgment* $\Gamma \vdash t : A$, written $\Gamma \models_{(\xi, \rho)} t : A$ if $(\!|t|\!)_\rho \in [\![A]\!]_\xi$.

2. A typing judgment $\Gamma \vdash t : A$ is *sound*, written $\Gamma \models t : A$, if for every valuation (ξ, ρ),

$$(\xi, \rho) \models \Gamma \Rightarrow \Gamma \models_{(\xi, \rho)} e.$$

We have:

THEOREM 29 (Soundness of Typing Judgments).

$$\Gamma \vdash t : A \Rightarrow \Gamma \models t : A.$$

Proof. By induction on the structure of derivations. The interesting cases are when the last rule applied is a (function) rule or a (recursion) rule:

1. (Function): Assume that the last step is

$$\frac{\Gamma \vdash \sigma\,\mathbf{A} : * \quad \Gamma \vdash t_i : \tau_i\,\mathbf{A} \quad 1 \leqslant i \leqslant n}{\Gamma \vdash f\,\mathbf{A}\,t_1 \cdots t_n : \sigma\,\mathbf{A}}.$$

Necessarily $f : (\tau_1, ..., \tau_n) \to \sigma$. Let $(\xi, \rho) \models \Gamma$. By the induction hypothesis, $(\!|t_i|\!)_\rho \in [\![\tau_i \mathbf{A}]\!]_\xi$ for $i = 1, ..., n$. By Definition 22, $f(\!|t_1|\!)_\rho \cdots (\!|t_n|\!)_\rho \in [\![\sigma \mathbf{A}]\!]_\xi$.

2. (Recursion): Assume that the last step is

$$\frac{\Gamma \vdash F : \mathscr{H}_r(I, J) \quad \forall r \in \mathscr{R}}{\Gamma \vdash \mathsf{rec}_\sigma^J[I]\,F : \sigma\,\mathbf{A} \to J_\sigma}.$$

Let $(\xi, \rho) \models \Gamma$. By the induction hypothesis, $(\!|F_r|\!)_\rho \in [\![\mathscr{H}_r(I, J)]\!]_\xi$. In particular, $(\!|F_r|\!)_\rho \in \mathsf{SN}$.

We prove by induction on the derivation of $M \in [\![\sigma I]\!]_\xi$ that $(\!|\mathsf{rec}_\sigma^J[I]\,F|\!)_\rho\,M \in [\![J_\sigma]\!]_\xi$:

(a) M is a base term. In that case, the induction step follows from the induction hypothesis.

(b) M is of the form $f\,t_1 \cdots t_n$. In that case, $t_i \in [\![\mathsf{maxdom}_r^\Sigma[i]\,I]\!]_\xi$ with $r = (\sigma, n, f) \in \mathscr{R}$ and $i = 1, ..., n$. Moreover,

$$\begin{aligned}(\!|\mathsf{rec}_\sigma^J[I]\,F|\!)_\rho\,M &= \mathsf{rec}_\sigma(\!|F|\!)_\rho\,(f\,t_1 \cdots t_n)\\ &\to_k (\!|F_r|\!)\,t_1 \cdots t_n\,G_{i_1} \cdots G_{i_k}\\ &= \mathbf{kred}((\!|\mathsf{rec}_\sigma^J[I]\,F|\!)_\rho\,M),\end{aligned}$$

where

i. $i_1 < \cdots < i_k$,

ii. for $1 \leqslant j \leqslant n$, $\mathsf{maxdom}_r^\Sigma[j] \in \Lambda \Leftrightarrow j \in \{i_1, ..., i_k\}$,

iii. for $1 \leqslant j \leqslant n$, $G_j = \mathsf{rec}_{\mathsf{maxdom}_r^\Sigma[j]}\,(\!|F|\!)_\rho\,t_j$.

By the induction hypothesis, $\mathsf{rec}_{\mathsf{maxdom}_r^\Sigma[j]}\,(\!|F|\!)_\rho\,t_j \in [\![J_{\mathsf{maxdom}_r^\Sigma[j]}]\!]_\xi$ for $j \in \{i_1, ..., i_k\}$. By assumption, $(\!|F_r|\!)_\rho \in [\![\mathscr{H}_r(I, J)]\!]_\xi$ and hence by definition of $\mathscr{H}$, $\mathbf{kred}((\!|\mathsf{rec}_\sigma^J[I]\,F|\!)_\rho\,M) \in [\![J_\sigma]\!]_\xi$. Saturated sets being closed under key-expansion, we conclude by noticing that $(\!|\mathsf{rec}_\sigma^J[I]\,F|\!)_\rho\,M \in \mathsf{SN}$.

(c) $M \to_k N$ and $N \in [\![\sigma I]\!]_\xi$ with a smaller derivation than $M \in [\![\sigma I]\!]_\xi$. In that case, we conclude by the induction hypothesis that $(\!|\mathsf{rec}_\sigma^J[I]\,F|\!)_\rho\,N \in [\![J_\sigma]\!]_\xi$. Now

$$(\!|\mathsf{rec}_\sigma^J[I]\,F|\!)_\rho\,N = \mathbf{kred}((\!|\mathsf{rec}_\sigma^J[I]\,F|\!)_\rho\,M).$$

Saturated sets being closed under key-expansion, we conclude by noticing that $(\!|\mathsf{rec}_\sigma^J[I]\,F|\!)_\rho\,M \in \mathsf{SN}$. ∎

COROLLARY 30 (Strong Normalization). *If $\Gamma \vdash t : A$, then all t is $\beta\iota$-strongly normalizing.*

Proof. Let Γ be an arbitrary context. Consider the valuation $(\xi_\Gamma, \rho_\Gamma)$, where $\xi_\Gamma(\alpha) = [\![A]\!]_{\xi_\Gamma}$ if $(\alpha \leqslant A : K) \in \Gamma$ and $\rho_\Gamma(x) = x$. Then $(\xi_\Gamma, \rho_\Gamma) \models \Gamma$. By Theorem 29, $|u| \in [\![A]\!]_{\xi_\Gamma}$ for every derivable typing judgment $\Gamma \vdash u : A$. By correctness of types, $\Gamma \vdash A : *$. By Proposition 26, $[\![A]\!]_{\xi_\Gamma} \in \mathbf{SAT}$. Hence $|t| \in [\![A]\!]_{\xi_\Gamma} \subseteq \mathsf{SN}$. We conclude from Lemma 16. ∎

5.3. Decidability of Type-Checking

Decidability of type checking is a fundamental property of typed λ-calculi. Indeed, program correctness in a typed programming language and proof-checking in a proof-development system are often reduced to type checking itself. Thus it is important to be able to type-check, i.e., to decide whether or not a judgment is derivable according to our type system. In this subsection, we give an algorithm for deciding whether a judgment $\Gamma \vdash t : A$ is derivable in $IF^\omega_\leqslant$. The algorithm, which is closely related to type-checking algorithms for $F^\omega_\leqslant$ [Com95, PS97], relies on:

1. an algorithm for deciding whether a kinding judgment is derivable;
2. an algorithm for deciding whether a subtyping judgment is derivable;
3. an algorithm for computing, when it exists, the minimal type of an object in a given context.

Throughout this subsection, we let M^{nf} denote, when it exists, the unique $\beta\iota$-normal form of an expression.

5.3.1. Decidability of Kinding

Kinding in $IF^\omega_\leqslant$ is shown to be decidable by giving a syntax-directed presentation of the kinding fragment of $IF^\omega_\leqslant$.

DEFINITION 31. The relation $\vdash_{\mathsf{sdk}}$ is defined in Table 9.

Note that $\vdash_{\mathsf{sdk}}$ is obtained from the kinding fragment of $\vdash$ simply by restricting the use of (weakening).

PROPOSITION 32. $\Gamma \vdash A : K \Leftrightarrow \Gamma \vdash_{\mathsf{sdk}} A : K$.

Proof. By induction on the structure of derivations. ∎

Decidability of kinding follows.

COROLLARY 33 (Decidability of Kinding). *It is decidable whether a kinding judgment is derivable.*

Proof. $\vdash_{\mathsf{sdk}}$ provides an algorithm for deciding. ∎

We conclude this section by defining a new relation to be used in the definition of algorithmic subtyping.

DEFINITION 34. $\Gamma \vdash_{\mathsf{sdknf}} A : K$ if $\Gamma \vdash_{\mathsf{sdk}} A : K$ and both Γ and A are in normal form.

TABLE 9

Syntax-Directed Rules for Kinding Judgments

(Bounded start)	$\dfrac{\Gamma \vdash_{\mathsf{sdk}} A : K}{\Gamma, \alpha \leqslant A : K \vdash_{\mathsf{sdk}} \alpha : K}$	if $\alpha \notin \Gamma$
(Weakening)	$\dfrac{\Gamma \vdash_{\mathsf{sdk}} B : K \quad \Gamma \vdash_{\mathsf{sdk}} A : *}{\Gamma, x : A \vdash_{\mathsf{sdk}} B : K}$	if $x \notin \Gamma$ and $B \in \mathbb{V}^{\Box} \cup \{\top^K\}$
(Bounded weakening)	$\dfrac{\Gamma \vdash_{\mathsf{sdk}} B : K' \quad \Gamma \vdash_{\mathsf{sdk}} A : K}{\Gamma, \alpha \leqslant A : K \vdash_{\mathsf{sdk}} B : K'}$	if $\alpha \notin \Gamma$ and $B \in \mathbb{V}^{\Box} \cup \{\top^K\}$
(Top)	$\dfrac{}{\vdash_{\mathsf{sdk}} \top_K : K}$	
(Product)	$\dfrac{\Gamma \vdash_{\mathsf{sdk}} A : * \quad \Gamma \vdash_{\mathsf{sdk}} B : *}{\Gamma \vdash_{\mathsf{sdk}} A \to B : *}$	
(Bounded product)	$\dfrac{\Gamma, \alpha \leqslant A : K \vdash_{\mathsf{sdk}} B : \circ}{\Gamma \vdash_{\mathsf{sdk}} \Pi \alpha \leqslant A : K . B : *}$	
(Application)	$\dfrac{\Gamma \vdash_{\mathsf{sdk}} A : K_1 \to K_2 \quad \Gamma \vdash_{\mathsf{sdk}} B : K_1}{\Gamma \vdash_{\mathsf{sdk}} AB : K_2}$	
(Abstraction)	$\dfrac{\Gamma, \alpha : K_1 \vdash_{\mathsf{sdk}} B : K_2}{\Gamma \vdash_{\mathsf{sdk}} \lambda \alpha : K_1 . B : K_1 \to K_2}$	
(Sort)	$\dfrac{\Gamma \vdash_{\mathsf{sdk}} A_q : * \quad 1 \leqslant q \leqslant \mathsf{size}(\mathscr{P})}{\Gamma \vdash_{\mathsf{sdk}} \sigma\, A_1 \cdots A_{\mathsf{size}(\mathscr{P})} : *}$	if $\sigma \in \Lambda$

5.3.2. *Decidability of Subtyping*

Decidability of subtyping, which constitutes the most intricate part of the proof of decidability of type checking, is typically established in four steps [Com95, PS97]:

1. give a syntax-directed presentation $\vdash_{\mathsf{alg}}$ of the subtyping system of $IF^{\omega}_{\leqslant}$;
2. prove that $\vdash_{\mathsf{alg}}$ is sound with respect to $\vdash$;
3. prove that $\vdash_{\mathsf{alg}}$ is complete with respect to $\vdash$;
4. show that $\vdash_{\mathsf{alg}}$ yields indeed a terminating algorithm.

For the sake of brevity, we do not treat the last step. Termination of the algorithm can be shown by extending Compagnoni's results [Com95].

DEFINITION 35 (Algorithmic Subtyping). 1. For every legal context Γ and $\alpha \in \mathbb{V}^{\Box}$, we let $\Gamma(\alpha)$ denote, if it exists, the unique $A \in \mathbb{C}$ s.t. $(\alpha \leqslant A : K) \in \Gamma$ for some $K \in \mathbb{K}$.

2. For every legal context Γ, $\alpha \in \mathbb{V}^{\Box}$ and $\mathbf{B} \in \mathbb{C}$, we let $\Gamma(\alpha\, \mathbf{B})$ denote $\Gamma(\alpha)\, \mathbf{B}$.
3. The relation $\vdash_{\mathsf{alg}}$ is defined by the rules of Table 10.
4. $\Gamma \vdash_{\mathsf{alg}} A \leqslant^* B$ if $\vdash_{\mathsf{alg}} A \leqslant B \wedge \Gamma \vdash A : * \; \Gamma \vdash B : *$.

TABLE 10

Rules for Algorithmic Subtyping

($\leqslant$-sort-alg)	$\dfrac{\Gamma \vdash_{\mathsf{alg}} A_q \leqslant^* A'_q \quad 1 \leqslant q \leqslant \mathsf{size}(\mathscr{P})}{\Gamma \vdash_{\mathsf{alg}} \sigma\,\mathbf{A} \leqslant \sigma'\,\mathbf{A}'}$	if $\sigma, \sigma' \in \Lambda$ and $\sigma \leqslant \sigma'$
($\leqslant$-start-alg)	$\dfrac{\Gamma \vdash_{\mathsf{alg}} A \leqslant A' \quad \Gamma \vdash_{\mathsf{sdk}} A : K}{\Gamma, \alpha \leqslant A : K \vdash_{\mathsf{alg}} \alpha \leqslant A'}$	if $\alpha \notin \Gamma$ and $A' \neq \top^K$
($\leqslant$-weakening-alg)	$\dfrac{\Gamma \vdash_{\mathsf{alg}} A \leqslant A' \quad \Gamma \vdash_{\mathsf{sdknf}} B : *}{\Gamma, x : B \vdash_{\mathsf{alg}} A \leqslant A'}$	if $x \notin \Gamma$, $A \in \mathbb{V}^{\square}$ $A \neq A'$ and $A' \neq \top^K$
($\leqslant$-bounded weakening-alg)	$\dfrac{\Gamma \vdash_{\mathsf{alg}} A \leqslant A' \quad \Gamma \vdash_{\mathsf{sdknf}} B : K}{\Gamma, \alpha \leqslant B : K \vdash_{\mathsf{alg}} A \leqslant A'}$	if $\alpha \notin \Gamma$, $A \in \mathbb{V}^{\square}$ $A \neq A'$ and $A' \neq \top^K$
($\leqslant$-top-alg)	$\dfrac{\Gamma \vdash_{\mathsf{sdknf}} A : K}{\Gamma \vdash_{\mathsf{alg}} A \leqslant \top^K}$	
($\leqslant$-refl-V-alg)	$\dfrac{\Gamma \vdash_{\mathsf{sdknf}} \alpha : K}{\Gamma \vdash_{\mathsf{alg}} \alpha \leqslant \alpha}$	if $\alpha \in \mathbb{V}^{\square}$
($\leqslant$-refl-A-agl)	$\dfrac{\Gamma \vdash_{\mathsf{sdknf}} AB : K}{\Gamma \vdash_{\mathsf{alg}} AB \leqslant AB}$	
($\leqslant$-product-alg)	$\dfrac{\Gamma \vdash_{\mathsf{alg}} A'_1 \leqslant^* A_1 \quad \Gamma \vdash_{\mathsf{alg}} A_2 \leqslant^* A'_2}{\Gamma \vdash_{\mathsf{alg}} A_1 \to A_2 \leqslant A'_1 \to A'_2}$	
($\leqslant$-bounded product-alg)	$\dfrac{\Gamma, \alpha \leqslant B : K \vdash_{\mathsf{alg}} A \leqslant^* A'}{\Gamma \vdash_{\mathsf{alg}} (\Pi\alpha \leqslant B : K.A) \leqslant (\Pi\alpha \leqslant B : K.A')}$	
($\leqslant$-application-alg)	$\dfrac{\Gamma \vdash_{\mathsf{alg}} (\Gamma(AB))^{\mathsf{nf}} \leqslant C \quad \Gamma \vdash_{\mathsf{sdknf}} AB : K}{\Gamma \vdash_{\mathsf{alg}} AB \leqslant C}$	if $AB \neq C$, $C \neq \top^K$

Note that all alg-derivable judgments are in normal form, in the sense that Γ, A, and B are in normal form whenever $\Gamma \vdash_{\mathsf{alg}} A \leqslant B$ is derivable.

PROPOSITION 36 (Soundness of $\vdash_{\mathsf{alg}}$ with Respect to $\vdash$). *For every context Γ and constructors A and B,*

$$\Gamma \vdash_{\mathsf{alg}} A \leqslant B \Rightarrow \Gamma \vdash A \leqslant B.$$

Proof. By induction on the structure of derivations. ∎

In order to prove the completeness of $\vdash_{\mathsf{alg}}$ with respect to $\vdash$, it is convenient to introduce a new derivation system $\vdash_{\mathsf{norm}}$ which extends $\vdash_{\mathsf{alg}}$.

DEFINITION 37. The relation $\vdash_{\mathsf{norm}}$ is defined by the rules of Table 10 and

$$(\leqslant\text{-reflexivity}) \quad \frac{\Gamma \vdash_{\mathsf{sdk}} A : K}{\Gamma \vdash_{\mathsf{norm}} A \leqslant A} \qquad \text{if } A \text{ is in normal form}$$

$$(\leqslant\text{-transivity}) \quad \frac{\Gamma \vdash_{\mathsf{norm}} A \leqslant A' \quad \Gamma \vdash_{\mathsf{norm}} A' \leqslant B}{\Gamma \vdash_{\mathsf{norm}} A \leqslant B}.$$

Then we proceed in two steps: first, we establish the equivalence between $\vdash_{\mathsf{alg}}$ and $\vdash_{\mathsf{norm}}$ (Proposition 38). Second, we prove the completeness of $\vdash_{\mathsf{norm}}$ with respect to $\vdash$ (Proposition 39).

PROPOSITION 38 (Equivalence between $\vdash_{\mathsf{alg}}$ and $\vdash_{\mathsf{norm}}$). *Assume that* $\Gamma \vdash A : K$ *and* $\Gamma \vdash B : K$. *Then*

$$\Gamma \vdash_{\mathsf{alg}} A \leqslant B \Leftrightarrow \Gamma \vdash_{\mathsf{norm}} A \leqslant B.$$

Proof. Only the right-to-left implication is interesting. The proof proceeds exactly as in [Com95].

1. Show that every norm-derivation can be reduced to a derivation where ($\leqslant$-reflexivity) is applied only to variables and applications (when $\mathscr{P} = \varnothing$, we would also need to apply reflexivity to sorts). Such derivation are called refl-normal.

2. Show that every refl-normal derivation with one application of ($\leqslant$-transitivity) can be reduced to an alg-derivation.

3. Conclude by (1) and successive applications of (2) that ($\leqslant$-reflexivity) and ($\leqslant$-transitivity) may be eliminated from every norm-derivation. ∎

PROPOSITION 39 (Completeness of $\vdash_{\mathsf{norm}}$ with Respect to $\vdash$). *Assume that* $\Gamma \vdash A : K$ *and* $\Gamma \vdash B : K$. *Then*

$$\Gamma \vdash A \leqslant B \Rightarrow \Gamma^{\mathsf{nf}} \vdash_{\mathsf{norm}} A^{\mathsf{nf}} \leqslant B^{\mathsf{nf}}.$$

Proof. By induction on the structure of derivations. The difficult induction step is when the rule considered is ($\leqslant$-application). In this case we use the fact that for every kind K, constructors A, B, and C, and context Γ,

$$(\Gamma \vdash AC : K \wedge \Gamma^{\mathsf{nf}} \vdash_{\mathsf{norm}} A^{\mathsf{nf}} \leqslant B^{\mathsf{nf}}) \Rightarrow \Gamma^{\mathsf{nf}} \vdash_{\mathsf{norm}} (AC)^{\mathsf{nf}} \leqslant (BC)^{\mathsf{nf}}.$$

Note that the fact itself is proved by induction on the structure of derivations. The proofs are easily adapted from [Com95]. ∎

As an easy consequence, we get decidability of subtyping.

PROPOSITION 40 (Decidability of Subtyping). *Assume that* $\leqslant$ *is decidable on sorts. Then it is decidable whether a typing judgment is derivable.*

Proof. Given a context Γ and two constructors A and B, we need to decide whether $\Gamma \vdash A \leqslant B$. Observe that $\Gamma \vdash A \leqslant B$ if and only if there exists $K \in \mathbb{K}$ such that both $\Gamma \vdash A : K$ and $\Gamma \vdash B : K$ are derivable and $\Gamma^{\mathsf{nf}} \vdash_{\mathsf{alg}} A^{\mathsf{nf}} \leqslant B^{\mathsf{nf}}$. Then conclude from termination of algorithmic subtyping. ∎

5.3.3. Decidability of Type Checking

Decidability of type checking is proved by giving a sound and complete type inference algorithm. As usual, we define a function $\mathsf{min}_\Gamma(t)$ which computes when it exists the minimal type of an object t in context Γ. Obviously, the existence of $\mathsf{min}_\Gamma(t)$ requires the underlying signature to satisfy some conditions.

DEFINITION 41 ([GM92]). A signature $\Sigma = (\mathscr{F}, \mathsf{decl})$ over $(\mathscr{P}, (\Lambda, \leqslant))$ is *preregular* if for every $f \in \mathscr{F}$ and $\boldsymbol{\sigma} \in (\Lambda \cup \mathscr{P})^\omega$ the set $\mathsf{Codom}^\Sigma_{(\boldsymbol{\sigma}, f)}$ is either empty or has a minimal element. If it exists, the minimal element of $\mathsf{codom}^\Sigma_{(\boldsymbol{\sigma}, f)}$ is denoted by $\mathsf{mincod}^\Sigma_{((\sigma_1, \ldots, \sigma_n), f)}$.

Pre-regularity, which holds for all the signatures considered in this paper, ensures that every Σ-term as a minimal sort. It also ensures that every object has a minimal type in a given context. In order to define the type inference algorithm, we need to specify an auxiliary function shape_Γ such that $\mathsf{shape}_\Gamma(A)$ is the smallest parameterized sort (i.e., expression of the form $\sigma\ \mathbf{B}$ with $\sigma \in \Lambda$), product, or functional type above A in context Γ.

DEFINITION 42. Let Γ be a context and $A \in \mathbb{C}$. Define

$$\mathsf{shape}_\Gamma(A) = \begin{cases} \mathsf{shape}_\Gamma(\Gamma(A^{\mathsf{nf}})) & \text{if} \quad A^{\mathsf{nf}} = \alpha\ \mathbf{C} \text{ and } \alpha \in \mathbb{V}^\square \\ A^{\mathsf{nf}} & \text{otherwise.} \end{cases}$$

Note that $\mathsf{shape}_\Gamma(A)$ is well defined provided that $\Gamma \vdash A : *$; the termination argument is similar to that of [Com95]. The following lemma captures the fundamental properties of shape_Γ.

LEMMA 43. *Assume that* $\Gamma \vdash A : *$.

1. $\Gamma \vdash A \leqslant \mathsf{shape}_\Gamma(A)$;
2. $\mathsf{shape}_\Gamma(A)$ *is a parametrized sort, a product, or a functional type*;
3. *if* $\Gamma \vdash A \leqslant B$ *and* B *is a parametrized sort, a product, or a functional type then* $\Gamma \vdash \mathsf{shape}_\Gamma(A) \leqslant B$.

We now define the type inference algorithm. For the remainder of this subsection, we assume that Σ is pre-regular. Moreover, we use $\Gamma \vdash_{\mathsf{alg}} A \leqslant B$ as an abbreviation for $\Gamma^{\mathsf{nf}} \vdash_{\mathsf{alg}} A^{\mathsf{nf}} \leqslant B^{\mathsf{nf}}$.

DEFINITION 44 (Type Inference). 1. The *type inference* relation $\vdash_{\mathsf{inf}}$ is defined by the rules of Table 11.

2. The *minimal typing* $\mathsf{min}_\Gamma(t)$ of an object t in a context Γ is the unique, if it exists, constructor A s.t. $\Gamma \vdash_{\mathsf{inf}} t : A$.

$\vdash_{\mathsf{inf}}$ is sound and complete with respect to $\vdash$ in the following sense:

PROPOSITION 45 (Minimal Typing).

$$\Gamma \vdash t : A \Leftrightarrow \Gamma \vdash \mathsf{min}_\Gamma(t) \leqslant A \wedge \Gamma \vdash_{\mathsf{inf}} t : \mathsf{min}_\Gamma(t).$$

Proof. By induction on the structure of derivations. For the right-to-left implication, it is enough to show that $\Gamma \vdash_{\mathsf{inf}} t : A$ implies $\Gamma \vdash t : A$. We prove the induction step of the left-to-right implication for the (function) rule. The (recursion) rule is treated in a similar manner. Assume that

$$\frac{\Gamma \vdash \sigma\ \mathbf{A} : * \quad \Gamma \vdash t_i : \tau_i\ \mathbf{A} \quad 1 \leqslant i \leqslant n}{\Gamma \vdash f\ \mathbf{A}\ t_1 \cdots t_n : \sigma\ \mathbf{A}}$$

TABLE 11

Rules for Type Inference

(Start)	$\dfrac{\Gamma \vdash_{\mathsf{sdk}} A : *}{\Gamma\, x : A \vdash_{\mathsf{inf}} x : A}$	if $x \notin \Gamma$ and $x \in \mathbb{V}^*$
(Weakening)	$\dfrac{\Gamma \vdash_{\mathsf{inf}} y : B \quad \Gamma \vdash_{\mathsf{sdk}} A : *}{\Gamma, x : A \vdash_{\mathsf{inf}} y : B}$	if $x \notin \Gamma$ and $y \in \mathbb{V}^*$
(Bounded weakening)	$\dfrac{\Gamma \vdash_{\mathsf{inf}} y : B \quad \Gamma \vdash_{\mathsf{sdk}} A : K}{\Gamma, \alpha \leqslant A : K \vdash_{\mathsf{inf}} y : B}$	if $\alpha \notin \Gamma$ and $y \in \mathbb{V}^*$
(Application)	$\dfrac{\Gamma \vdash_{\mathsf{inf}} t : C \quad \Gamma \vdash_{\mathsf{inf}} u : A' \quad \Gamma \vdash_{\mathsf{alg}} A' \leqslant A}{\Gamma \vdash_{\mathsf{inf}} tu : B}$	if $\mathsf{shape}_\Gamma(C) = A \to B$
(Bounded application)	$\dfrac{\Gamma \vdash_{\mathsf{inf}} t : C \quad \Gamma \vdash_{\mathsf{alg}} A' \leqslant A}{\Gamma \vdash_{\mathsf{inf}} tA' : B[A'/\alpha]}$	if $\mathsf{shape}_\Gamma(C) = \Pi\alpha \leqslant A : K\,.\,B$
(Abstraction)	$\dfrac{\Gamma, x : A \vdash_{\mathsf{inf}} t : B}{\Gamma \vdash_{\mathsf{inf}} \lambda x : A\,.\,P : A \to B}$	
(Bounded abstraction)	$\dfrac{\Gamma, \alpha \leqslant A : K \vdash_{\mathsf{inf}} t : B}{\Gamma \vdash_{\mathsf{inf}} \lambda\alpha \leqslant A : K.t : \Pi\alpha \leqslant A : K.B}$	
(Function)	$\dfrac{\Gamma \vdash_{\mathsf{sdk}} \sigma\, \mathbf{A} : * \quad \Gamma \vdash_{\mathsf{inf}} t_i : B_i \quad \Gamma \vdash_{\mathsf{alg}} \mathsf{shape}_\Gamma(B_i) \leqslant \sigma_i\, \mathbf{A} \quad 1 \leqslant i \leqslant n}{\Gamma \vdash_{\mathsf{inf}} h\, \mathbf{A}\, t_1 \cdots t_n : \tau\, \mathbf{A}}$	if $\mathsf{shape}_\Gamma(B_i) = \sigma_i\, \mathbf{A}_i$ and $\tau = \mathsf{mincod}_{((\sigma_1, \ldots, \sigma_n), h)}$
(Recursion)	$\dfrac{\Gamma \vdash_{\mathsf{inf}} F_r : C_r \quad \Gamma \vdash_{\mathsf{alg}} C_r \leqslant (\mathscr{H}_r(I, J)) \quad \forall r \in \mathscr{R}}{\Gamma \vdash_{\mathsf{inf}} \mathsf{rec}^J_\sigma[I]\, F : \sigma\, I \to J_\sigma}$	

with f: $(\tau_1, \ldots, \tau_n) \to \sigma$. Set $C_i = \mathsf{min}_\Gamma(t_i)$. By the induction hypothesis,

$$\Gamma \vdash_{\mathsf{inf}} t_i : C_i$$
$$\Gamma \vdash C_i \leqslant \tau_i\, \mathbf{A}$$

for $1 \leqslant i \leqslant n$.

To prove $\Gamma \vdash_{\mathsf{inf}} f\, \mathbf{A}\, t_1 \cdots t_n : \mathsf{min}_\Gamma(f\, \mathbf{A}\, t_1 \cdots t_n)$. Necessarily $\Gamma \vdash \mathsf{shape}_\Gamma(C_i) \leqslant \tau_i\, \mathbf{A}$ and hence by (function),

$$\Gamma \vdash_{\mathsf{inf}} f\, \mathbf{A}\, t_1 \cdots t_n : \mathsf{mincod}^{\Sigma}_{((\tau'_1, \ldots, \tau'_n), f)}\, \mathbf{A},$$

so we are done.

To prove $\Gamma \vdash \mathsf{min}_\Gamma(f\, \mathbf{A}\, t_1 \cdots t_n) \leqslant \sigma\, \mathbf{A}$. Necessarily $\mathsf{shape}_\Gamma(C_i) = \tau'_i\, \mathbf{A}_i$ with $\tau'_i \leqslant \tau_i$ and $\Gamma \vdash A^i_q \leqslant A_q$ for $1 \leqslant q \leqslant \mathsf{size}(\mathscr{P})$ and $1 \leqslant i \leqslant n$. Moreover $\mathsf{mincod}^{\Sigma}_{((\tau'_1, \ldots, \tau'_n), f)} \leqslant \sigma$ and hence

$$\Gamma \vdash \mathsf{mincod}^{\Sigma}_{((\tau'_1, \ldots, \tau'_n), f)}\, \mathbf{A} \leqslant \sigma\, \mathbf{A},$$

so we are done. ▮

COROLLARY 46 (Decidability of Typing). *It is decidable whether a typing judgment is derivable.*

Proof. We need to show that $\vdash_{\mathsf{inf}}$ terminates. It is an immediate consequence of the termination of the subtype-checking algorithm $\vdash_{\mathsf{alg}}$, of the kinding algorithm $\vdash_{\mathsf{sdk}}$, and of the fact that the side conditions can be checked. ▮

5.4. Subject Reduction and Consistency

Kinding is preserved under reduction.

LEMMA 47 (Constructor Subject Reduction). *If* $\Gamma \vdash A : K$ *and* $A \to_\beta A'$, *then* $\Gamma \vdash A' : K$.

Proof. By induction on the structure of derivations. ▮

Similarly, typing is preserved under reduction. However, the proof of object subject seduction is not trivial: it requires some key lemmas which ensure that subtyping is "structurally defined." Typically, the key lemmas state properties of the form

$$A \to B \leqslant A' \to B' \Rightarrow (A' \leqslant A \wedge B \leqslant B').$$

However, these key lemmas are hard to establish because of the ($\leqslant$-transitivity) rule. Fortunately, we dispose of the equivalence between normal and algorithmic subtyping to conclude.

LEMMA 48 (Object Subject Reduction). *If* $\Gamma \vdash v : C$ *and* $v \to_{\beta\iota} v'$, *then* $\Gamma \vdash v' : C$.

Proof. Show the key lemmas:

1. $\Gamma \vdash A \to B \leqslant A' \to B' \Rightarrow (\Gamma \vdash A' \leqslant A \wedge \Gamma \vdash B \leqslant B')$.
2. $\Gamma \vdash (\Pi\alpha \leqslant A : K \cdot B) \leqslant (\Pi\alpha \leqslant A' : K' \cdot B') \Rightarrow (K = K' \wedge A' =_{\beta_\omega} A \wedge \Gamma, \quad \alpha \leqslant A : K \vdash B \leqslant B')$.

Then proceed by induction on the structure of derivations. In the case of the (application) rule, one needs to use (1) for $M = (\lambda x : A \cdot t)\, u$ and (2) for $M = (\lambda\alpha \leqslant A : K \cdot t)\, B$. ▮

Subject reduction, together with normalization, implies that the calculus is consistent.

PROPOSITION 49 (Consistency). *$IF^\omega_{\leqslant}$ is consistent*; *i.e.*, *there is no* $t \in \mathbb{O}$ *such that* $\vdash t : (\Pi\alpha : * \cdot \alpha)$.

Proof. If there were such a t, by normalization and subject reduction there would be a u in $\beta\iota$-normal form such that $\vdash u : (\Pi\alpha : * \cdot \alpha)$. Proceed by a case analysis on the possible structure of u to show that such a u cannot exist. ▮

6. CONCLUSION

This paper focuses on the interaction between subtyping and recursion in the context of a typed λ-calculus with subtyping. Its main contribution is to show that the interaction can be controlled in a satisfactory manner provided that some mild restrictions are imposed on order-sorted signatures.

6.1. Related Work

Inductive and recursive definitions for order-sorted data types have been studied in a series of papers [DOB98, KD98, OD91] by Owe, Dahl, Bastiansen, and Kristoffersen. These works are carried out in the context of ABEL, a specification language developed at Oslo University. Their work emphasizes the expressibility of the framework and suggests a paradigm, called terminating generator induction (TGI), which provides a pattern-matching-like facility for recursive definitions. However, they do not address issues such as strong normalization or decidability of type checking, which form the subject of this paper. In addition to the above-mentioned works, Bastiansen [Bas95] has recently studied the use of parametric subtypes in ABEL, but this work contains limited information concerning TGI in this context.

Our work also shares some motivations with refinement types [FP91, Pfe93], which are used by Pfenning to encode various formal languages in an extension of the Logical Frameworks. However, the technicalities are rather different, in that we use overloading instead of intersection types and in that we introduce recursion operators for data types.

In addition, our work is directly related to the general area of inductive types, see, e.g., [Dyb94, PM93], and subtyping, see, e.g., [AC96b, Com95, PS97, Ste97].

6.2. Directions for Future Work

This work studies some aspects of the interaction between inductive types and subtyping but many issues remain to be investigated.

1. The system is not well behaved with respect to canonical inhabitants; e.g., $\mathsf{nil}\ \mathsf{even}$ is a closed normal inhabitant of $\mathsf{List}\ \mathsf{nat}$ in a system with a parametric data type of lists and a type of odd and even natural numbers. From a proof-theoretical perspective, it seems important to address this anomaly, which is pervasive in type systems with subtyping.

2. For practical applications, it seems important to allow for definitions by pattern matching, see, e.g., [Coq92, Cor97, Elb98], and for overloaded definitions (at the moment, it is not possible to define the predecessor function pred with types

$$\mathsf{nat} \to \mathsf{nat}$$

$$\mathsf{odd} \to \mathsf{even}.$$

Instead one can define

$$\mathsf{pred}_{\mathsf{nat}} : \mathsf{nat} \to \mathsf{nat}$$

$$\mathsf{pred}_{\mathsf{odd}} : \mathsf{odd} \to \mathsf{even}$$

but this solution leads to clutter in large examples).

3. The extension of Pure Type Systems [Bar92] with (first-order, strictly overloaded) order-sorted inductive types could provide a suitable framework for

proof-development systems, in particular for formalizing natural semantics [Kah87]. It would be interesting to study this framework.

4. A treatment of subtyping for inductive families (e.g., having $X_n \leqslant X_{n+1}$, where X_n is the inductive type with n-elements; see [NPS90, Chap. 6]) would provide a framework in which to formalize typed languages with subtyping, including the various typed object calculi considered in [AC96a].

Finally it would be interesting to combine the approach suggested in this paper with other approaches to subtyping; see, e.g., [Bet98, Luo98].

ACKNOWLEDGMENTS

The author is grateful to the anonymous referees for their remarks and suggestions. Earlier versions of the paper were written while the author was working at the Center for Mathematics and Computer Science (CWI), Amsterdam, and Chalmers University, Göteborg. The author acknowledges financial support from the Dutch Science Foundation (NWO) and from the European TMR programme.

Received April 19, 1996; final manuscript received July 30, 1998

REFERENCES

[AC96a] Abadi, M., and Cardelli, L. (1996), "A Theory of Objects," Springer-Verlag, Berlin/New York.

[AC96b] Aspinall, D., and Compagnoni, A. (1996), Subtyping dependent types, *in* "Proceedings, LICS'96," pp. 86–97, IEEE Computer Society Press, Los Alamitos, CA.

[Acz77] Aczel, P. (1977), An introduction to inductive definitions, *in* "Handbook of Mathematical Logic" (J. Bacwise, Ed.), Vol. 90, Studies in Logic and the Foundations of Mathematics, pp. 739–782, North-Holland, Amsterdam.

[Bar92] Barendregt, H. (1992), Lambda calculi with types, *in* "Handbook of Logic in Computer Science" (S. Abramsky, D. Gabay, and T. Maibaum, Eds.), Vol. 2, pp. 117–309, Oxford Science Publications, London.

[Bas95] Bastiansen, T. J. (1995), "Parametric Subtypes in ABEL," Technical Report 207, Department of Informatics, University of Oslo.

[Bet98] Betarte, G. (1998), "Dependent Record Types and Algebraic Structures in Type Theory," Ph.D. thesis, Department of Computer Science, Chalmers Tekniska Högskola.

[Car90] Cardelli, L. (1990), Notes about $F^{\omega}_{\leqslant}$, manuscript, D.E.C..

[Com95] Compagnoni, A. (1995), "Higher-Order Subtyping with Intersection Types," Ph.D. thesis, Department of Computer Science, University of Nijmegen.

[Coq92] Coquand, T. (1992), Pattern matching in type theory, *in* "Informal Proceedings of Logical Frameworks'92" (B. Nordström, Ed.), pp. 66–79.

[Cor97] Cornes, C. (1997), "Conception d'un langage de haut niveau de représentation de preuves: Récurrence par filtrage de motifs; Unification en présence de types inductifs primitifs; Synthèse de lemmes d'inversion," Ph.D. thesis, Université de Paris 7.

[DOB98] Dahl, O.-J., Owe, O., and Bastiansen, T. J. (1998), Subtyping and constructive specification, *Nordic J. Computing* **5**(1).

[Dyb94] Dybjer, P. (1994), Inductive families, *Formal Aspects Comput.* **6**, 440–465.

[Elb98] Elbers, H. (1998), "Connecting Formal and Informal Mathematics," Ph.D. thesis, Technische Universiteit, Eindhoven.

[FP91] Freeman, T., and Pfenning, F. (Refinement types for ML), *in* "Proceedings, PLDI'91," pp. 268–277, Assoc. Comput. Mach., New York.

[GD94] Goguen, J., and Diaconescu, R. (1994), An Oxford survey of order sorted algebra, *Math. Struct. Comput. Sci.* **4**, 363–392.

[GM85] Goguen, J., and Meseguer, J. (1985), Completeness of many-sorted equational logic, *Houston J. Math.* **11**(3), 307–334.

[GM92] Goguen, J., and Meseguer, J. (1992), Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations, *Theoretic. Comput. Sci.* **105**(2), 216–273.

[Hal93] Hallgren, T. (1993), "Subtypes in Polymorphic Functional Languages," Licenciate thesis, Department of Computer Science, Chalmers Tekniska Högskola.

[HP95] Hofmann, M., and Pierce, B. (1995), A unifying type-theoretic framework for objects, *J. Funct. Programming* **5**(4), 593–635.

[Kah87] Kahn, G. (1987), Natural semantics, *in* "Proceedings, Symposium on Theoretical Aspects of Computer Science," pp. 22–39, Lecture Notes in Computer Science, Vol. 247, Springer-Verlag, Berlin/New York.

[KD98] Kristroffersen, B., and Dahl, O.-J. (1998), On introducing higher order functions in ABEL, *Nordic J. Comput.* **5**(1).

[KOR93] Klop, J. W., van Oostrom, V., and van Raamsdonk, F. (1993), Combinatory reduction systems: Introduction and survey, *Theoret. Comput. Sci.* **121**(1–2), 279–308.

[Luo98] Luo, Z. (Coercive subtyping), *J. Logic Comput.*, to appear.

[MG85] Meseguer, J., and Goguen, J. (1985), Initiality, induction and computability, *in* "Algebraic Methods in Semantics" (M. Nivat and J. Reynolds, Eds.), pp. 459–541, Cambridge Univ. Press, Cambridge, UK.

[NPS90] Nordström, B., Petersson, K., and Smith, J. (1990), "Programming in Martin-Löf's Type Theory. An Introduction," International Series of Monographs on Computer Science, Vol. 7, Oxford Univ. Press, London.

[OD91] Owe, O., and Dahl, O.-J. (1991), Generator induction in order-sorted algebras, *Formal Aspects Comput.* **3**(1), 2–20.

[Pfe93] Pfenning, F. (1993), Refinement types for logical frameworks, *in* "Informal Proceedings of TYPES'93" (H. Geuvers, Ed.), pp. 285–299.

[PM93] Paulin-Mohring, C. (1993), Inductive definitions in the system Coq. Rules and properties, *in* "Proceedings, TLCA'93" (M. Bezem and J. F. Groote, Eds.), Lectures Notes in Computer Science, pp. 328–345, Vol. 664, Springer-Verlag, Berlin/New York.

[Pol94] Pollack, R. (1994), "The Theory of LEGO: A Proof Checker for the Extended Calculus of Constructions," Ph.D. thesis, University of Edinburgh.

[PS97] Pierce, B., and Steffen, M. (1997), Higher-order subtyping, *Theoret. Comput. Sci.* **176**(1–2), 253–282.

[PT94] Pierce, B., and Turner, D. (1994), Simple type-theoretic foundations for object-oriented programming, *J. Funct. Programming* **4**(2), 207–247.

[Ste97] Steffen, M. (1997), "Polarized Higher-Order Subtyping," Ph.D. thesis, Department of Computer Science, University of Erlangen.